

emCrypt

Cryptographic algorithm library

User Guide & Reference Manual

Document: UM12006
Software Version: 2.36
Revision: 0
Date: June 1, 2022



A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2014-2021 SEGGER Microcontroller GmbH, Monheim am Rhein / Germany

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel. +49 2173-99312-0
Fax. +49 2173-99312-28
E-mail: support@segger.com*
Internet: www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at <https://www.segger.com/legal/privacy-policy/>.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: June 1, 2022

Software	Revision	Date	By	Description
2.36	0	210422	PC	Chapter "Hash algorithms" <ul style="list-style-type: none"> • Added BLAKE2b and BLAKE2s algorithms. Chapter "Symmetric encryption" <ul style="list-style-type: none"> • Added PRESENT algorithm. Chapter "MAC algorithms" <ul style="list-style-type: none"> • Added CMAC-PRESENT algorithm.
2.34.2	0	210420	PC	Update to latest software version.
2.34.1	0	190808	PC	Update to latest software version.
2.34	0	190510	PC	Chapter "Symmetric encryption" <ul style="list-style-type: none"> • Added IDEA algorithm. Chapter "MAC algorithms" <ul style="list-style-type: none"> • Added CMAC-IDEA algorithm. • Added Michael algorithm.
2.32	0	190311	PC	Update to latest software version.
2.30	0	181010	PC	Chapter "MAC algorithms" <ul style="list-style-type: none"> • Added AES-XCBC-MAC algorithm. • Added Camellia-XCBC-MAC algorithm. • Added ARIA-XCBC-MAC algorithm. • Added SEED-XCBC-MAC algorithm. • Added Twofish-XCBC-MAC algorithm.
2.28	0	180928	PC	Chapter "Hash algorithms" <ul style="list-style-type: none"> • Added SM3 algorithm.
2.26	0	180621	PC	Chapter "Storage device encryption" <ul style="list-style-type: none"> • Added XTS-AES algorithm. • Added XTS-ARIA algorithm. • Added XTS-Camellia algorithm. • Added XTS-SEED algorithm. • Added XTS-Twofish algorithm. Chapter "Symmetric encryption" <ul style="list-style-type: none"> • Added ChaCha20 algorithm. • Added RC4 algorithm. Chapter "Hash algorithms" <ul style="list-style-type: none"> • Added support for SHA-224 hardware acceleration. Chapter "Extendable-output functions" <ul style="list-style-type: none"> • Added Keccak algorithm. Chapter "Configuring emCrypt" <ul style="list-style-type: none"> • Added CRYPTO_OS_Request(). Chapter "Hardware acceleration" <ul style="list-style-type: none"> • Added "STM32 AES coprocessor (Add-on)". • Added "STM32 HASH coprocessor (Add-on)".
2.24	0	171116	PC	Chapter "MAC algorithms" <ul style="list-style-type: none"> • Added Poly1305-AES algorithm. • Added Poly1305-ARIA algorithm. • Added Poly1305-SEED algorithm. • Added Poly1305-Camellia algorithm. • Added Poly1305-Twofish algorithm. Chapter "Hash algorithms" <ul style="list-style-type: none"> • Added "...IsInstalled" methods. Chapter "Symmetric encryption" <ul style="list-style-type: none"> • Added "...IsInstalled" methods.
2.22	0	170823	PC	Chapter "Component API" <ul style="list-style-type: none"> • Added CRYPTO_GetVersionText(). • Added CRYPTO_GetCopyrightText(). Chapter "Hash algorithms" <ul style="list-style-type: none"> • All type-safe APIs conform to identical profiles. Chapter "MAC algorithms" <ul style="list-style-type: none"> • Added Poly1305 algorithm. Chapter "Key encapsulation" <ul style="list-style-type: none"> • Added RSAES-OAEP decryption functions.

Software	Revision	Date	By	Description
				Chapter "Hardware acceleration" • Added.
2.20	0	170728	PC	Chapter "Ciphers" • Added Shoup 8-bit table optimization. Chapter "Extendable-output functions" • Added incremental functions. Chapter "EdDSA" • Added Ed448 signature scheme. • Enhanced Ed25519 signature scheme.
2.10	0	170601	PC	Added configuration to chapters: • "MD5", "RIPEMD160" • "SHA-1", "SHA-256", "SHA-512" • "AES", "DES", "ARIA", "SEED" • "Camellia", "Blowfish", "Twofish"
2.00	0	170425	RH	Added introduction to chapters: • "Hash algorithms" • "MAC algorithms" • "Symmetric encryption" • "RSA encryption" • "RSA signatures" • "ECDSA signatures" Added sections: • "Dynamic memory usage" • "Initializing emCrypt"
1.18	0	170419	PC	Prerelease, with • X25519 scalar multiplication. • Curve25519 field arithmetic shared between X25519 and Ed25519.
1.16	0	170412	PC	Prerelease, with • Configurable twin multiply for ECDSA signature verification. • Windowing for point multiplication. • Implementation of Brainpool curves. • Runtime configuration of MPI allocation unit size.
1.14	0	170406	PC	Prerelease, with • ECDSA CAVS KATs. • RSA key generation CAVS KATs.
1.12	0	170330	PC	Prerelease, with • GMAC. • GHASH.
1.10	0	170323	PC	First prerelease.
1.00	0	151001	PC	Internal release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0-13-1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
Parameter	Parameters in API functions.
Sample	Sample code in program examples.
Sample comment	Comments in program examples.
User Input	Text entered at the keyboard by a user in a session transcript.
Secret Input	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

1	Introduction	12
1.1	What is emCrypt?	13
1.2	Target audience	14
1.3	Package content	15
1.4	Sample applications	16
1.5	Naming conventions	18
1.6	API conventions	20
1.7	Design considerations	21
1.8	Dynamic memory usage	22
2	Building emCrypt	23
2.1	Quick start	24
2.2	Installing CMake	25
2.3	Unpacking and configuring	26
3	Configuring emCrypt	29
3.1	Initializing emCrypt	30
3.2	CRYPTO-OS integration	31
4	Component API	41
4.1	Preprocessor symbols	42
4.2	API functions	43
5	Hash algorithms	47
5.1	Introduction	48
5.2	BLAKE2b	49
5.3	BLAKE2s	71
5.4	MD5	92
5.5	RIPEMD-160	117
5.6	SHA-1	142
5.7	SHA-224	169
5.8	SHA-256	190
5.9	SHA-384	216
5.10	SHA-512	234
5.11	SHA-512/224	260
5.12	SHA-512/256	276
5.13	SHA3-224	292
5.14	SHA3-256	314
5.15	SHA3-384	336

5.16	SHA3-512	358
5.17	SM3	380
5.18	GHASH	402
6	MAC algorithms	409
6.1	Introduction	411
6.2	CMAC-AES	412
6.3	CMAC-TDES	431
6.4	CMAC-IDEA	450
6.5	CMAC-CAST	467
6.6	CMAC-SEED	483
6.7	CMAC-ARIA	500
6.8	CMAC-Camellia	517
6.9	CMAC-Twofish	534
6.10	CMAC-Blowfish	551
6.11	CMAC-PRESENT	567
6.12	GMAC-AES	584
6.13	GMAC-SEED	599
6.14	GMAC-ARIA	614
6.15	GMAC-Camellia	629
6.16	GMAC-Twofish	644
6.17	HMAC-MD5	659
6.18	HMAC-RIPEMD-160	678
6.19	HMAC-SHA-1	696
6.20	HMAC-SHA-224	718
6.21	HMAC-SHA-256	738
6.22	HMAC-SHA-384	759
6.23	HMAC-SHA-512	779
6.24	HMAC-SHA-512/224	799
6.25	HMAC-SHA-512/256	817
6.26	HMAC-SHA3-224	835
6.27	HMAC-SHA3-256	852
6.28	HMAC-SHA3-384	869
6.29	HMAC-SHA3-512	886
6.30	HMAC-SM3	902
6.31	XCBC-AES	920
6.32	XCBC-SEED	940
6.33	XCBC-ARIA	958
6.34	XCBC-Camellia	976
6.35	XCBC-Twofish	994
6.36	KMAC	1012
6.37	Poly1305	1021
6.38	Poly1305-AES	1033
6.39	Poly1305-SEED	1052
6.40	Poly1305-ARIA	1069
6.41	Poly1305-Camellia	1086
6.42	Poly1305-Twofish	1103
6.43	Michael	1120
7	Symmetric encryption (secret key)	1138
7.1	Introduction	1139
7.2	DES	1140
7.3	AES	1190
7.4	IDEA	1254
7.5	SEED	1293
7.6	ARIA	1345
7.7	Camellia	1397
7.8	CAST	1450
7.9	ChaCha20	1494

7.10	Blowfish	1508
7.11	Twofish	1552
7.12	PRESENT	1604
7.13	RC4	1646
7.14	Building blocks	1651
8	Storage device encryption	1665
8.1	XTS-AES	1666
8.2	XTS-ARIA	1669
8.3	XTS-Camellia	1672
8.4	XTS-SEED	1675
8.5	XTS-Twofish	1678
9	Random bit generation	1681
9.1	Fortuna	1682
9.2	Hash-DRBG-SHA-1	1692
9.3	Hash-DRBG-SHA-224	1698
9.4	Hash-DRBG-SHA-256	1704
9.5	Hash-DRBG-SHA-384	1710
9.6	Hash-DRBG-SHA-512	1716
9.7	Hash-DRBG-SHA-512/224	1722
9.8	Hash-DRBG-SHA-512/256	1728
9.9	HMAC-DRBG-SHA-1	1734
9.10	HMAC-DRBG-SHA-224	1740
9.11	HMAC-DRBG-SHA-256	1746
9.12	HMAC-DRBG-SHA-384	1752
9.13	HMAC-DRBG-SHA-512	1758
9.14	HMAC-DRBG-SHA-512/224	1764
9.15	HMAC-DRBG-SHA-512/256	1770
9.16	CTR-DRBG-TDES	1776
9.17	CTR-DRBG-AES-128	1782
9.18	CTR-DRBG-AES-192	1788
9.19	CTR-DRBG-AES-256	1794
10	Key derivation	1800
10.1	KDF1	1801
10.2	KDF2	1831
10.3	X9.63 KDF	1861
10.4	HKDF	1882
10.5	PBKDF2	1916
11	Extendable-output functions	1927
11.1	SHAKE128	1928
11.2	SHAKE256	1936
11.3	cSHAKE	1944
11.4	Keccak	1955
12	Asymmetric encryption (public key)	1961
12.1	RSA	1962
13	Digital signatures	1989
13.1	RSA	1990
13.2	DSA	2108
13.3	ECDSA	2169
13.4	EdDSA	2251

14	Key encapsulation	2303
14.1	RSAES-OAEP	2304
14.2	RSAES-PKCS1	2329
14.3	AES-KW	2331
14.4	SEED-KW	2340
14.5	ARIA-KW	2347
14.6	Camellia-KW	2354
14.7	Twofish-KW	2361
15	Key agreement	2368
15.1	Overview	2369
15.2	Diffie-Hellman key agreement	2370
15.3	Elliptic curve Diffie-Hellman key agreement	2379
16	Elliptic curves	2393
16.1	Overview	2394
16.2	Data types	2395
16.3	Predefined curves	2398
16.4	Arithmetic	2399
16.5	Self-test API	2443
17	Multiprecision integers	2447
17.1	Management	2448
17.2	Assignment	2455
17.3	Comparisons and predicates	2464
17.4	Addition and subtraction	2485
17.5	Multiplication	2505
17.6	Division	2518
17.7	Exponentiation	2535
17.8	Bit and byte-level access	2578
17.9	Logical operations	2595
17.10	Algorithms	2597
17.11	Random numbers	2603
17.12	Format conversion	2608
17.13	Primes	2620
18	Cryptographic message syntax	2632
18.1	CCM and GCM	2633
19	Hardware acceleration	2636
19.1	EFM32 CRYPTO coprocessor (Add-on)	2637
19.2	Kinetis CAU coprocessor (Add-on)	2651
19.3	LPC18S and LPC43S AES ROM (Add-on)	2654
19.4	iMX RT10xx data coprocessor (Add-on)	2656
19.5	STM32 AES coprocessor (Add-on)	2659
19.6	STM32 CRYP coprocessor (Add-on)	2660
19.7	STM32 HASH coprocessor (Add-on)	2663
20	Utilities	2666
20.1	Buffer manipulation	2667
20.2	TLV parsing	2714
20.3	Counters	2748
21	Benchmarks	2753
21.1	Ciphers	2754
21.2	Hashing	2769

21.3	Public key	2778
21.4	Random bits	2802
22	Resource use	2807
22.1	Context sizes	2808
23	Bibliography	2811
23.1	IETF RFCs	2811
23.2	ANSI standards	2811
23.3	ISO standards	2811
23.4	IEEE standards	2811
23.5	NIST standards and special publications	2812
23.6	Other relevant documents	2812
24	Glossary	2813
25	Indexes	2816
25.1	Index of functions	2817
25.2	Subject index	2847

Chapter 1

Introduction

This manual describes the interfaces made available by emCrypt to the application programmer.

1.1 What is emCrypt?

emCrypt is practical cryptographic algorithm library that is designed to run on embedded systems. It is designed to be small, efficient, secure, and broad enough to function as the basis of security protocols such as SSL, SSH, and IPsec. emCrypt is the foundation of all SEGGER security products — emSSL, emSSH, emSecure-RSA, emSecure-ECDSA — and is shared between them.

emCrypt is not a library of algorithms for research into cryptography, it does not target absolute performance with complex algorithms requiring large working stores, nor does it offer every hashing and ciphering scheme ever devised and found through Google. It does not offer the general ability to mix algorithms and modes to construct encryption schemes that are of little practical use. Should you require this, then emCrypt is not for you. emCrypt targets what is needed for industry-standard protocols, and to do this with robust, cleanly-engineered code. If you absolutely require some scheme that we do not support, you can always ask us to devote some engineering time to the problem.

emCrypt has the capability to use hardware accelerators, if they are available, to accelerate ciphering, hashing, and public key cryptography. SEGGER have written support for several popular embedded cryptographic accelerators so customers can immediately put these to use in end applications.

1.2 Target audience

This manual is a reference for the emCrypt cryptographic library. It is not intended as a tutorial on security, nor will it help you design secure protocols. Therefore, we assume that you are familiar with cryptographic principles and simply need to know how to put emCrypt to use and, optionally, gain an insight into the underlying implementation techniques.

1.3 Package content

emCrypt is provided in source code and the exact content depends upon the versions and add-ons that you purchase. The following table shows the content of the package:

Files	Description
Config	Configuration header files.
CRYPTO	emCrypt cryptographic library source code.
Doc	emCrypt documentation.
Sample/Config	Example emCrypt user configuration.
SEGGER	SEGGER software component source code.
Application	emCrypt sample applications.

1.4 Sample applications

The emCrypt library ships with a number of sample applications that demonstrate how to integrate IoT capability into your application. Each sample application demonstrates a specific capability of the emCrypt library or is a small incremental step over previous examples.

1.4.1 Benchmark samples

The sample applications are:

Application	Description
CRYPTO_Bench_AES.c	Benchmark AES performance.
CRYPTO_Bench_DES.c	Benchmark DES and TDES performance.
CRYPTO_Bench_Camellia.c	Benchmark Camellia performance.
CRYPTO_Bench_ECDH.c	Benchmark ECDH key agreement performance.
CRYPTO_Bench_ECDSA.c	Benchmark ECDSA sign and verify performance.
CRYPTO_Bench_Hashes.c	Benchmark performance of all hash algorithms.
CRYPTO_Bench_MD5.c	Benchmark MD5 performance.
CRYPTO_Bench_ModExp.c	Benchmark performance of all modular exponentiation algorithms by implementation.
CRYPTO_Bench_RIPEMD160.c	Benchmark RIPEMD-160 performance.
CRYPTO_Bench_RNG.c	Benchmark performance of all DRBG algorithms.
CRYPTO_Bench_SHA1.c	Benchmark SHA-1 performance.
CRYPTO_Bench_SHA256.c	Benchmark SHA-256 performance.
CRYPTO_Bench_SHA512.c	Benchmark SHA-512 performance.
CRYPTO_Bench_SHA3.c	Benchmark SHA-3 performance.

1.4.2 Self-test samples

The sample applications are:

Application	Description
CRYPTO_Test_All.c	Run all algorithm self-tests.
CRYPTO_Test_AES.c	Run AES self-tests.
CRYPTO_Test_DES.c	Run DES self-tests.
CRYPTO_Test_SEED.c	Run SEED self-tests.
CRYPTO_Test_ARIA.c	Run ARIA self-tests.
CRYPTO_Test_Camellia.c	Run Camellia self-tests.
CRYPTO_Test_MD5.c	Run MD5 self-tests.
CRYPTO_Test_RIPEMD160c	Run RIPEMD-160 self-tests.
CRYPTO_Test_SHA1.c	Run SHA-1 self-tests.
CRYPTO_Test_SHA256.c	Run SHA-256 self-tests.
CRYPTO_Test_SHA512.c	Run SHA-512 self-tests.
CRYPTO_Test_EdDSA.c	Run Ed25519 self-tests.

1.4.3 Other samples

The sample applications are:

Application	Description
CRYPTO_DumpContextSize.c	Display all algorithm context sizes.

1.5 Naming conventions

emCrypt uses a number of naming conventions for functions, types, variables, and preprocessor symbols. These conventions are described in this section.

1.5.1 Product namespace

All emCrypt functions, types, variables, and preprocessor symbols are prefixed by `CRYPTO` to indicate they are part of the emCrypt product and to prevent name clashes with other libraries.

1.5.2 Abstract interfaces (APIs)

An emCrypt API is a generic interface to a set of data and functions that implement that interface. The API is defined as a C structure grouping data members and function pointers and can be viewed as a C++ abstract class or as a Java interface.

The name of the interface, as a C type, is of the following form:

```
CRYPTO_name_API
```

The `CRYPTO` prefix defines the namespace as above. The suffix `API` indicates that the type is an emCrypt API.

emCrypt has the following abstract APIs:

API name	Description
<code>CRYPTO_RNG_API</code>	Interface for random numbers.
<code>CRYPTO_CIPHER_API</code>	Interface for ciphers.
<code>CRYPTO_HASH_API</code>	Interface for message digest algorithms.
<code>CRYPTO_MAC_API</code>	Interface for message authentication code algorithms.
<code>CRYPTO_MODEXP_API</code>	Interface for modular exponentiation algorithms.

emCrypt offers concrete implementations conforming to these APIs.

1.5.3 Functions conforming to an API

A function that conforms to a function prototype in an API places the name of the API immediately following the `CRYPTO` prefix:

```
CRYPTO_api-name_...
```

As an example, the function that initializes an AES cipher in encryption mode and that conforms to the `CIPHER API` is:

```
void CRYPTO_CIPHER_AES_InitEncrypt(void *pSelf, const U8 *pKey, unsigned KeyLen);
```

1.5.4 Functions accepting fixed-size data

In some cases there are two implementations of a function where both do essentially the same work. One implementation takes a *length* parameter and the other does not. When the length can be implied from the context, it is not necessary to pass the length as a parameter.

For instance, initializing an AES cipher in encryption mode is a matter of calling the following function:

```
void CRYPTO_CIPHER_AES_InitEncrypt(void *pSelf, const U8 *pKey, unsigned KeyLen);
```

In many cases the key length is known in advance, for instance when initializing AES encryption with a 128-bit key (AES-128). In this case, emCrypt offers an additional function that provides this capability:

```
void CRYPTO_CIPHER_AES_128_InitEncrypt(void *pSelf, const U8 *pKey);
```

This drops the key length and places it where it is commonly expected, in this case after the "AES".

This convention is applied consistently throughout emCrypt. For instance, even though the name for 128-bit KMAC is standardized as KMAC128 by NIST, emCrypt uses KMAC_128 separating the key length and algorithm.

1.5.5 Functions delivering fixed-size data

Following on from the previous section, there are functions that typically deliver fixed-size data but are also required to deliver truncated data by some algorithms. A MAC or hash is an example of this and, in the same way as the key size above, two (or more) functions are provided.

The first delivers a MAC with the possibility of truncation:

```
void CRYPTO_MAC_HMAC_SHA1_Final(void *pSelf, U8 *pMAC, unsigned MACLen);
```

And the remainder deliver MACs of different (fixed) sizes:

```
void CRYPTO_MAC_HMAC_SHA1_Final_160(void *pSelf, U8 *pMAC);
void CRYPTO_MAC_HMAC_SHA1_Final_96 (void *pSelf, U8 *pMAC);
```

In this case the size of the data delivered is placed at the end of the function name. The MAC functions above deliver 160 bits of data (a full HMAC-SHA-1 MAC) and a 96-bit truncated MAC (HMAC-SHA-1-96).

The emCrypt convention is that all *output size* information is placed at the end of the function name even though the algorithm name (HMAC-SHA-1-96) would suggest that it should come after SHA1 and before Final.

1.5.6 Self-test names

The general naming convention is:

```
CYPTO_algorithm[_mode]_source_SelfTest()
```

The *algorithm* refers to the algorithm under test (e.g. AES) or a particular group of functions (e.g. MPI, multi-precision integer arithmetic).

The *mode* is something such as signature (Sign), signature verification (Verify), a cipher mode (e.g. GCM or CCM), or is omitted if the self-test combines everything required to test the module as a unit (e.g. a symmetric cipher).

The *source* describes the source of the test vectors, for instance a standards document, a web page, or something else recognizable. For test vectors that originate from NIST as part of the CAVS suite, they would be named with "CAVS" as the source. EMC are a source of some vectors, RFCs are sources of other vectors, and others are taken from specifications with associated test vectors available on the Internet.

1.6 API conventions

1.6.1 Parameter order

All functions that operate on an algorithm context always pass the algorithm context as the first parameter.

All function that require a memory allocator context always pass the context as the final parameter.

Output parameters always precede input parameters.

1.6.2 Null pointer inputs

Unless otherwise documented, all parameters that take a pointer to an object require that the pointer be nonnull. If a null pointer is acceptable to a function, it is documented as being acceptable in the **Parameter** section or in the **Additional Information** section if there are special or complex conditions for acceptability.

A special case is made for *compound parameters* where an address and a size that define an object are passed to a function: if the size is zero, the address may be the null pointer.

1.7 Design considerations

1.7.1 Multithreading and reentrancy

All algorithmic functions are designed to be reentrant. For those that take a context, such as an encryption context, hash context, memory allocation context and so on, reentrancy is guaranteed only if each context in the two (or more) threads of execution is different.

Sharing contexts between different functions requires a mutual exclusion mechanism to protect the context. This mechanism is left to the user to implement. Although possible, it is recommended that memory allocators *do not* implement mutual exclusion themselves as this leads to suboptimal performance in multithreaded systems—it is much more efficient to ensure mutual exclusion above the emCrypt API at the application level.

1.8 Dynamic memory usage

Some of the functions of emCrypt use data objects that may grow during operation, for example the multi precision integers needed for asymmetric cryptography. The caller has to provide a memory context (of type CRYPTO_MEM_CONTEXT) to all of these functions. The memory context has to be initialized before it can be used. This requires a memory allocator and a memory buffer of fixed size, that will be used to store the dynamic objects. Segger provides several memory allocators for this purpose that are shipped with emCrypt. The memory context may be initialized globally for the whole application or locally to perform only a few cryptographic operations. It may be discarded if the objects stored in it are not used any more.

Example

```
//  
// Example using SEGGER_MEM_SIMPLE_HEAP.  
//  
int Sign(const U8 *pData, U32 DataLen, U8 *pResult) {  
    int r;  
    SEGGER_MEM_SIMPLE_HEAP SimpleHeap;  
    SEGGER_MEM_CONTEXT      MemContext;  
    U32                      BigBuff[1024];  
  
    //  
    // Initialize memory context.  
    //  
    SEGGER_MEM_SIMPLE_HEAP_Init(&MemContext, &SimpleHeap,  
                                &BigBuff[0], sizeof(BigBuff), 8);  
    //  
    // Perform cryptographic operation.  
    //  
    r = CRYPTO_RSA_PKCS1_SHA1_Sign(&PrivateKey, pData, DataLen,  
                                 NULL, 0, pResult, MAX_SIZE, &MemContext);  
    //  
    // Memory context is discarded upon return of the function.  
    //  
    return r;  
}
```

Chapter 2

Building emCrypt

This section describes how to build emCrypt on Windows and Linux.

2.1 Quick start

emCrypt is distributed with a CMake file that enables you to build the demonstration emCrypt files on Windows and Linux to get up and running quickly. This section describes how to use CMake to build these examples using Visual Studio on Windows and using the standard `make` utility on Linux.

2.2 Installing CMake

Before you can build emCrypt, you must install CMake 2.8 or later. You can find CMake distributions for Windows on the CMake.org download page, <https://cmake.org/download/>.

The distributed software, and this section, are accurate using CMake 3.5.2.

For Linux, you can usually find and install precompiled versions of CMake using whatever software installation tool comes with your particular distribution.

2.3 Unpacking and configuring

2.3.1 Building on Windows

Once you can unzipped your application into a *clean directory*, you will see a number of subdirectories and a top-level file called CMakeLists.txt.

```
C:> dir

Directory of C:\Work

23/03/2017 21:53    <DIR>      .
23/03/2017 21:53    <DIR>      ..
23/03/2017 21:53    <DIR>      Application
23/03/2017 21:38            1,931 CMakeLists.txt
23/03/2017 21:53    <DIR>      Config
23/03/2017 21:53    <DIR>      CRYPTO
23/03/2017 21:53    <DIR>      Doc
23/03/2017 21:53    <DIR>      Sample
23/03/2017 21:53    <DIR>      SEGGER
23/03/2017 21:53    <DIR>      Windows

C:> _
```

Typically, to keep directories from becoming polluted with build outputs and temporary files, CMake users create an *out-of-source build directory* that keeps their image clean:

```
C:> mkdir Build
C:> cd Build
C:> _
```

Once in the build directory, it's time to configure the application using CMake:

```
C:> cmake . .
-- Building for: Visual Studio 14 2015
-- The C compiler identification is MSVC 19.0.24215.1
-- The CXX compiler identification is MSVC 19.0.24215.1
-- Check for working C compiler using: Visual Studio 14 2015
-- Check for working C compiler using: Visual Studio 14 2015 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler using: Visual Studio 14 2015
-- Check for working CXX compiler using: Visual Studio 14 2015 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Work/Build

C:> _
```

In the build directory you will find a Visual Studio solution file that you can open:

```
C:> dir *.sln

23/03/2017 21:59           33,984 emCrypt.sln

C:> _
```

You should now be able to build all the sample applications, and the emCrypt library, from within the Visual Studio IDE.

2.3.2 Building on Linux

Using Linux to build emCrypt and the sample applications is not very different from Windows. Create a Build directory for the out-of-source build and configure using CMake:

```
paul@ubuntu:~/Work/emCrypt mkdir Build
paul@ubuntu:~/Work/emCrypt/Build cd Build
paul@ubuntu:~/Work/emCrypt/Build cmake ...
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Build files have been written to: /home/paul/Work/emCrypt/Build

paul@ubuntu:~/Work/emCrypt/Build _
```

All you have to do now is use the standard `make` utility to build:

```
paul@ubuntu:~/Work/emCrypt/Build make
-- Build files have been written to: /home/paul/Work/emCrypt/Build
Scanning dependencies of target SEGGER
[ 0%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_SYS_IO_Linux.c.o
[ 1%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_SYS_Linux.c.o
[ 1%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_SYS_OS_Linux.c.o
[ 1%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_MEM.c.o
[ 1%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_memxor.c.o
[ 2%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_MEM_CHUNK_HEAP.c.o
[ 2%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_MEM_SBUFFER.c.o
[ 2%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_MEM_SIMPLE_HEAP.c.o
[ 2%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_MEM_SYSTEM_HEAP.c.o
[ 2%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_SYS.c.o
[ 3%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_SYS_IO.c.o
[ 3%] Building C object CMakeFiles/SEGGER.dir/SEGGER/SEGGER_VERSION.c.o
[ 3%] Linking C static library libSEGGER.a
[ 3%] Built target SEGGER
Scanning dependencies of target CRYPTO
[ 3%] Building C object CMakeFiles/CRYPTO.dir/CRYPTO/CRYPTO_AES.c.o
[ 4%] Building C object CMakeFiles/CRYPTO.dir/CRYPTO/
CRYPTO_AES_128_CAVS_SelfTest.c.
...
[ 99%] Building C object CMakeFiles/CRYPTO_TestAES.dir/Application/
CRYPTO_TestAES.c.o
[100%] Linking C executable CRYPTO_TestAES
[100%] Built target CRYPTO_TestAES
Scanning dependencies of target CRYPTO_TestCamellia
[100%] Building C object CMakeFiles/CRYPTO_TestCamellia.dir/Application/
CRYPTO_TestCamellia.c.o
[100%] Linking C executable CRYPTO_TestCamellia
[100%] Built target CRYPTO_TestCamellia
paul@ubuntu:~/Work/emCrypt/Build _
```

The applications are built into the Build directory for you to run:

```
paul@ubuntu:~/Work/emCrypt/Build ./CRYPTO_Test_AES
```

Copyright (c) 2014-2018 SEGGER Microcontroller GmbH www.segger.com

```
AES Self-Test compiled Mar 18 2018 16:31:03

Algorithm      Source    Status #Test
-----
AES-128-ECB    RFC 3602  PASS   2
AES-128-ECB    CAVS     PASS   568
AES-128-CCM    CAVS     PASS   720
AES-128-GCM    CAVS     PASS   7875
AES-192-ECB    CAVS     PASS   700
AES-192-CCM    CAVS     PASS   720
AES-192-GCM    CAVS     PASS   7875
AES-256-ECB    CAVS     PASS   810
AES-256-CCM    CAVS     PASS   720
AES-256-GCM    CAVS     PASS   7875
AES-CCM        SP800-38C PASS   12
```

All tests passed.

```
paul@ubuntu:~/Work/emCrypt/Build _
```

Chapter 3

Configuring emCrypt

3.1 Initializing emCrypt

Before using any emCrypt service you must initialize the CRYPTO module. You do this by including the emCrypt header `CRYPTO.h` and by calling `CRYPTO_Init()`.

```
//  
// Initialize emCrypt.  
//  
CRYPTO_Init();
```

You configure the capabilities of emCrypt in the function `CRYPTO_X_Config()` that is called as part of the emCrypt initialization carried out by `CRYPTO_Init`. `CRYPTO_X_Config()` must be provided in your application as a function with external linkage and an example is shipped with emCrypt.

Sample implementations of `CRYPTO_X_Config()` can be found in *Crypto-OS binding for embOS* on page 36 and *Crypto-OS binding for bare metal* on page 39.

Additionally the functions `CRYPTO_OS_Init()`, `CRYPTO_OS_Claim()`, `CRYPTO_OS_Request()` and `CRYPTO_OS_Unclaim()` must be provided by the application. If hardware acceleration is used in a threaded execution environment, these functions are required to lock hardware resources against simultaneously access by different threads, see *Crypto-OS integration* on page 31. Otherwise the functions may be empty as provided in file `CRYPTO_OS_None.c` from the emCrypt shipping.

3.2 CRYPTO-OS integration

In a threaded execution environment individual hardware resources must be protected from simultaneous use by more than one thread. emCrypt does this by surrounding use of hardware resources by calls to an OS binding layer.

To use a shared resource, emCrypt will either:

- Call `CRYPTO_OS_Claim()`, use the resource, and call `CRYPTO_OS_Unclaim()` to release it, or
- Call `CRYPTO_OS_Request()` to request access to the resource. If access is granted, emCrypt uses the resource and then calls `CRYPTO_OS_Unclaim()` to release it. In the case where access to the resource is not granted, emCrypt will not use the resource and will not call `CRYPTO_OS_Unclaim()`.

The parameter `Unit` is a zero-based index to the hardware being requested and is defined by the specific hardware platform or target device that is in use. No hardware acceleration interface in emCrypt requires more than three units (e.g. a ciphering unit, a hashing unit, and a random number generation unit). The specific requirements for each device are described in the relevant sections.

As an OS layer may well need to create mutexes or semaphores corresponding to each unit, `CRYPTO_OS_Init()` is called as part of emCrypt initialization.

3.2.1 CRYPTO-OS API

Function	Description
<code>CRYPTO_OS_Init()</code>	Initialize CRYPTO binding to OS.
<code>CRYPTO_OS_Claim()</code>	Claim a hardware resource.
<code>CRYPTO_OS_Request()</code>	Test-and-claim a hardware resource.
<code>CRYPTO_OS_Unclaim()</code>	Release claim on a hardware resource.

3.2.1.1 CRYPTO_OS_Init()

Description

Initialize CRYPTO binding to OS.

Prototype

```
void CRYPTO_OS_Init(void);
```

Additional information

This function should initialize any semaphores or mutexes used for protecting each hardware unit.

3.2.1.2 CRYPTO_OS_Claim()

Description

Claim a hardware resource.

Prototype

```
void CRYPTO_OS_Claim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Claim the hardware resource that corresponds to the unit index. In a threaded environment, this function should block a task requesting a resource that is already in use by using a semaphore or mutex, for example. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function can be empty.

3.2.1.3 CRYPTO_OS_Request()

Description

Test-and-claim a hardware resource.

Prototype

```
int CRYPTO_OS_Request(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Return value

= 0 Resource is already in use and was not claimed.
≠ 0 Resource claimed.

Additional information

Attempt to claim the hardware resource that corresponds to the unit index. In a threaded environment, this function is a nonblocking test-and-lock of a semaphore or mutex. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function should always return nonzero, i.e. resource claimed.

3.2.1.4 CRYPTO_OS_Unclaim()

Description

Release claim on a hardware resource.

Prototype

```
void CRYPTO_OS_Unclaim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Release the claim the hardware resource that corresponds to the unit index. This will only be called to unclaim a claimed resource.

3.2.2 CRYPTO-OS binding for embOS

The following is a sample binding for SEGGER embOS, CRYPTO_OS_embOS.c:

```
*****
*          (c) SEGGER Microcontroller GmbH & Co. KG
*          The Embedded Experts
*          www.segger.com
*****  

----- END-OF-HEADER -----  

File      : CRYPTO_OS_embOS.c
Purpose   : SEGGER embOS CRYPTO-OS binding.  

*/  

/*****  

*  

*      #include section  

*  

*****  

*/  

#include "CRYPTO.h"
#include "RTOS.h"  

/*****  

*  

*      Preprocessor definitions, configurable  

*  

*****  

*/  

#ifndef CRYPTO_CONFIG_OS_MAX_UNIT
#define CRYPTO_CONFIG_OS_MAX_UNIT      (CRYPTO_OS_MAX_INTERNAL_UNIT + 3)
#endif  

/*****  

*  

*      Static data  

*  

*****  

*/  

static OS_SEMAPHORE _aSema[CRYPTO_CONFIG_OS_MAX_UNIT];  

/*****  

*  

*      Public functions  

*  

*****  

*/  

/*****  

*  

*      CRYPTO_OS_Claim()  

*  

*      Function description  

*      Claim a hardware resource.  

*  

*      Parameters  

*      Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Claim(unsigned Unit) {
    if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
        OS_Error(OS_ERR_HW_NOT_AVAILABLE);
```

```

    }
    //
    OS_WaitCSema(&_aSema[Unit]);
}

/*****
*
*      CRYPTO_OS_Request()
*
*  Function description
*      Request a hardware resource.
*
*  Parameters
*      Unit - Zero-based index to hardware resource.
*
*  Return value
*      == 0 - Resource is already in use and was not claimed.
*      != 0 - Resource claimed.
*/
int CRYPTO_OS_Request(unsigned Unit) {
    if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
        OS_Error(OS_ERR_HW_NOT_AVAILABLE);
    }
    //
    return OS_CSemaRequest(&_aSema[Unit]);
}

/*****
*
*      CRYPTO_OS_Unclaim()
*
*  Function description
*      Release claim on a hardware resource.
*
*  Parameters
*      Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
    if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
        OS_Error(OS_ERR_HW_NOT_AVAILABLE);
    }
    //
    OS_SignalCSema(&_aSema[Unit]);
}

/*****
*
*      CRYPTO_OS_Init()
*
*  Function description
*      Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
    unsigned Unit;
    //
    for (Unit = 0; Unit < CRYPTO_CONFIG_OS_MAX_UNIT; ++Unit) {
        OS_CreateCSema(&_aSema[Unit], 1);
    }
}

/*****
*
*      CRYPTO_OS_Exit()
*
*  Function description
*      Deinitialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Exit(void) {

```

```
unsigned Unit;
//
for (Unit = 0; Unit < CRYPTO_CONFIG_OS_MAX_UNIT; ++Unit) {
    OS_DeleteCSema(&_aSema[Unit]);
}
*****
***** End of file *****
```

3.2.3 CRYPTO-OS binding for bare metal

The following is a sample binding for a bare metal system that has no tasking, CRYPTO_OS_None.c:

```
/*
 *          (c) SEGGER Microcontroller GmbH
 *          The Embedded Experts
 *          www.segger.com
 */

----- END-OF-HEADER -----


File      : CRYPTO_OS_None.c
Purpose   : Bare metal CRYPTO-OS binding.

*/



#include "CRYPTO.h"



/*
 *      Public code
 *
*****



*/
void CRYPTO_OS_Claim()
{
    * Function description
    *     Claim a hardware resource.
    *
    * Parameters
    *     Unit - Zero-based index to hardware resource.
    */
    void CRYPTO_USE_PARA(Unit);
}

/*
 *      CRYPTO_OS_Request()
 *
 *      Function description
 *      Test-and-claim a hardware resource.
 *
 *      Parameters
 *      Unit - Zero-based index to hardware resource.
 *
 *      Return value
 *      == 0 - Resource is already in use and was not claimed.
 *      != 0 - Resource claimed.
 */
int CRYPTO_OS_Request(unsigned Unit)
{
    CRYPTO_USE_PARA(Unit);
    return 1;
}

/*
 *      CRYPTO_OS_Unclaim()
 *
 *      Function description
 *      Release claim on a hardware resource.
 */
```

```
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
    CRYPTO_USE_PARA(Unit);
}

*****
*
*      CRYPTO_OS_Init()
*
*  Function description
*    Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
    /* Nothing to do. */
}

*****
*
*      CRYPTO_OS_Init()
*
*  Function description
*    Deinitialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Exit(void) {
    /* Nothing to do. */
}

***** End of file *****
```

Chapter 4

Component API

This chapter describes the API functions that related to the emCrypt component as a whole.

4.1 Preprocessor symbols

4.1.1 Version number

Description

Symbol expands to a number that identifies the specific emCrypt release.

Definition

```
#define CRYPTO_VERSION    23800
```

Symbols

Definition	Description
CRYPTO_VERSION	Format is "Mmmrr" so, for example, 12304 corresponds to version 1.23d.

4.2 API functions

The following table lists the component API functions.

Function	Description
<code>CRYPTO_GetCopyrightText()</code>	Get copyright as printable string.
<code>CRYPTO_GetVersionText()</code>	Get version as printable string.
<code>CRYPTO_Init()</code>	Initialize CRYPTO component.

4.2.1 CRYPTO_GetCopyrightText()

Description

Get copyright as printable string.

Prototype

```
char *CRYPTO_GetCopyrightText(void);
```

Return value

Zero-terminated copyright string.

4.2.2 CRYPTO_GetVersionText()

Description

Get version as printable string.

Prototype

```
char *CRYPTO_GetVersionText(void);
```

Return value

Zero-terminated version string.

4.2.3 CRYPTO_Init()

Description

Initialize CRYPTO component.

Prototype

```
void CRYPTO_Init(void);
```

Chapter 5

Hash algorithms

emCrypt implements the following message digest algorithms:

- BLAKE2
- MD5
- RIPEMD-160
- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512
- SHA-512/224
- SHA-512/256
- SHA3-224
- SHA3-256
- SHA3-384
- SHA3-512
- SM3

5.1 Introduction

In general a hash calculation is performed in three steps:

- Initialising the calculation.
- Processing input data. This step can be repeated multiple times.
- Calculating the final hash value.

The intermediate results are stored in a data structure called a 'hash context'. The hash context is maintained by the hash functions, only the memory must be provided by the caller. It can be discarded after the final hash calculation is done.

The API functions are named in the same way for all hash algorithms:

- `CRYPTO_<hash_name>_Init()` for initializing.
- `CRYPTO_<hash_name>_Add()` to process data.
- `CRYPTO_<hash_name>_Final()` to calculate the final hash value.

Example

```
//  
// Example for a SHA-1 hash calculation.  
//  
CRYPTO_SHA1_CONTEXT SHAContext;  
U8 aDigest[CRYPTO_SHA1_DIGEST_BYTE_COUNT];  
//  
// Initialize the hash context.  
//  
CRYPTO_SHA1_Init(&SHAContext);  
//  
// Process input data.  
//  
CRYPTO_SHA1_Add(&SHAContext, Data1, Data1Len);  
//  
// More data.  
//  
CRYPTO_SHA1_Add(&SHAContext, Data2, Data2Len);  
//  
// Calculate hash.  
//  
CRYPTO_SHA1_Final(&SHAContext, aDigest, sizeof(aDigest));  
//  
// aDigest now contains the hash value.  
// From now, SHAContext is not used any more.  
//
```

For every hash algorithm there is also a function to perform the whole hash calculation in one step. These functions are called `CRYPTO_<hash_name>_Calc()` and provide an easy way to calculate a hash from a single piece of data.

Besides the type-safe API functions described above, there are also generic API functions, that use a `void` pointer to take the hash context. These are useful, if the API functions shall be called via functions pointers to dynamically choose different hash algorithms. When using the generic functions the caller is responsible to provide the correct context (or memory areas) via the `void` pointer argument.

5.2 BLAKE2b

5.2.1 Standards reference

BLAKE2b is specified by the following document:

- IETF RFC 7693 — *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*

5.2.2 Algorithm parameters

5.2.2.1 Block size

```
#define CRYPTO_BLAKE2B_BLOCK_BYTE_COUNT 128
```

The number of bytes in a single BLAKE2B block.

5.2.2.2 Digest size

```
#define CRYPTO_BLAKE2B_DIGEST_BIT_COUNT 512
#define CRYPTO_BLAKE2B_DIGEST_BYTE_COUNT 64
```

The number of bits and bytes required to hold a complete BLAKE2b digest.

5.2.3 Type-safe API

The following table lists the BLAKE2b type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_BLAKE2B_Calc()</code>	Calculate digest.
<code>CRYPTO_BLAKE2B_Calc_512()</code>	Calculate digest, fixed size.
Incremental functions	
<code>CRYPTO_BLAKE2B_Init()</code>	Initialize context.
<code>CRYPTO_BLAKE2B_InitEx()</code>	Initialize context, extended.
<code>CRYPTO_BLAKE2B_Add()</code>	Add data to digest.
<code>CRYPTO_BLAKE2B_Get()</code>	Get incremental digest.
<code>CRYPTO_BLAKE2B_Final()</code>	Finalize digest calculation.
<code>CRYPTO_BLAKE2B_Final_512()</code>	Finalize digest calculation, fixed size.
<code>CRYPTO_BLAKE2B_Kill()</code>	Destroy context.
Setup and hardware acceleration	
<code>CRYPTO_BLAKE2B_Install()</code>	Install BLAKE2b hash implementation.
<code>CRYPTO_BLAKE2B_IsInstalled()</code>	Query whether hash algorithm is installed.
<code>CRYPTO_BLAKE2B_QueryInstall()</code>	Query BLAKE2b hardware accelerator.

5.2.3.1 CRYPTO_BLAKE2B_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_BLAKE2B_Add(      CRYPTO_BLAKE2B_CONTEXT * pSelf,
                           const U8          * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.2.3.2 CRYPTO_BLAKE2B_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_BLAKE2B_Calc(      U8      * pOutput,
                               unsigned   OutputLen,
                           const U8      * pInput,
                               unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.2.3.3 CRYPTO_BLAKE2B_Calc_512()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_BLAKE2B_Calc_512(      U8      * pOutput,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 64 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.2.3.4 CRYPTO_BLAKE2B_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_BLAKE2B_Final(CRYPTO_BLAKE2B_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.2.3.5 CRYPTO_BLAKE2B_Final_512()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_BLAKE2B_Final_512(CRYPTO_BLAKE2B_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 64 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.2.3.6 CRYPTO_BLAKE2B_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_BLAKE2B_Get(CRYPTO_BLAKE2B_CONTEXT * pSelf,  
                         U8                      * pOutput,  
                         unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.2.3.7 CRYPTO_BLAKE2B_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_BLAKE2B_Init(CRYPTO_BLAKE2B_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.2.3.8 CRYPTO_BLAKE2B_Install()

Description

Install BLAKE2b hash implementation.

Prototype

```
void CRYPTO_BLAKE2B_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.2.3.9 CRYPTO_BLAKE2B_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_BLAKE2B_IsInstalled(void);
```

Return value

- = 0 Hash algorithm is not installed.
- ≠ 0 Hash algorithm is installed.

5.2.3.10 CRYPTO_BLAKE2B_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_BLAKE2B_Kill(CRYPTO_BLAKE2B_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.2.3.11 CRYPTO_BLAKE2B_QueryInstall()

Description

Query BLAKE2b hardware accelerator.

Prototype

```
void CRYPTO_BLAKE2B_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                  const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.2.4 Generic API

The following table lists the BLAKE2b functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_BLAKE2B_Init()</code>	Initialize context.
<code>CRYPTO_HASH_BLAKE2B_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_BLAKE2B_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_BLAKE2B_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_BLAKE2B_Kill()</code>	Destroy digest.

5.2.4.1 CRYPTO_HASH_BLAKE2B_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_BLAKE2B_Add(      void      * pContext,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.2.4.2 CRYPTO_HASH_BLAKE2B_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_BLAKE2B_Final(void * pContext,  
                                U8      * pDigest,  
                                unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.2.4.3 CRYPTO_HASH_BLAKE2B_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_BLAKE2B_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.2.4.4 CRYPTO_HASH_BLAKE2B_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_BLAKE2B_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.2.4.5 CRYPTO_HASH_BLAKE2B_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_BLAKE2B_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.2.5 Self-test API

The following table lists the BLAKE2b self-test API functions.

Function	Description
CRYPTO_BLAKE2B_RFC7693_SelfTest()	Run BLAKE2 KATs from RFC 7693.
CRYPTO_BLAKE2B_Ref_SelfTest()	Run BLAKE2b reference self-tests.

5.2.5.1 CRYPTO_BLAKE2B_RFC7693_SelfTest()

Description

Run BLAKE2 KATs from RFC 7693.

Prototype

```
void CRYPTO_BLAKE2B_RFC7693_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.2.5.2 CRYPTO_BLAKE2B_Ref_SelfTest()

Description

Run BLAKE2b reference self-tests.

Prototype

```
void CRYPTO_BLAKE2B_Ref_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.3 BLAKE2s

5.3.1 Standards reference

BLAKE2s is specified by the following document:

- IETF RFC 7693 — *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*

5.3.2 Algorithm parameters

5.3.2.1 Block size

```
#define CRYPTO_BLAKE2S_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single BLAKE2S block.

5.3.2.2 Digest size

```
#define CRYPTO_BLAKE2S_DIGEST_BIT_COUNT 256  
#define CRYPTO_BLAKE2S_DIGEST_BYTE_COUNT 32
```

The number of bits and bytes required to hold a complete BLAKE2s digest.

5.3.3 Type-safe API

The following table lists the BLAKE2s type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_BLAKE2S_Calc()</code>	Calculate digest.
<code>CRYPTO_BLAKE2S_Calc_256()</code>	Calculate digest, fixed size.
Incremental functions	
<code>CRYPTO_BLAKE2S_Init()</code>	Initialize context.
<code>CRYPTO_BLAKE2S_InitEx()</code>	Initialize context, extended.
<code>CRYPTO_BLAKE2S_Add()</code>	Add data to digest.
<code>CRYPTO_BLAKE2S_Get()</code>	Get incremental digest.
<code>CRYPTO_BLAKE2S_Final()</code>	Finalize digest calculation.
<code>CRYPTO_BLAKE2S_Final_256()</code>	Finalize digest calculation, fixed size.
<code>CRYPTO_BLAKE2S_Kill()</code>	Destroy context.
Setup and hardware acceleration	
<code>CRYPTO_BLAKE2S_Install()</code>	Install BLAKE2s hash implementation.
<code>CRYPTO_BLAKE2S_IsInstalled()</code>	Query whether hash algorithm is installed.
<code>CRYPTO_BLAKE2S_QueryInstall()</code>	Query BLAKE2s hardware accelerator.

5.3.3.1 CRYPTO_BLAKE2S_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_BLAKE2S_Add(      CRYPTO_BLAKE2S_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.3.3.2 CRYPTO_BLAKE2S_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_BLAKE2S_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8   * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.3.3.3 CRYPTO_BLAKE2S_Calc_256()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_BLAKE2S_Calc_256(      U8      * pOutput,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 32 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.3.3.4 CRYPTO_BLAKE2S_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_BLAKE2S_Final(CRYPTO_BLAKE2S_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.3.3.5 CRYPTO_BLAKE2S_Final_256()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_BLAKE2S_Final_256(CRYPTO_BLAKE2S_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.3.3.6 CRYPTO_BLAKE2S_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_BLAKE2S_Get(CRYPTO_BLAKE2S_CONTEXT * pSelf,  
                         U8                      * pOutput,  
                         unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.3.3.7 CRYPTO_BLAKE2S_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_BLAKE2S_Init(CRYPTO_BLAKE2S_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.3.3.8 CRYPTO_BLAKE2S_Install()

Description

Install BLAKE2s hash implementation.

Prototype

```
void CRYPTO_BLAKE2S_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.3.3.9 CRYPTO_BLAKE2S_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_BLAKE2S_IsInstalled(void);
```

Return value

- = 0 Hash algorithm is not installed.
- ≠ 0 Hash algorithm is installed.

5.3.3.10 CRYPTO_BLAKE2S_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_BLAKE2S_Kill(CRYPTO_BLAKE2S_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.3.3.11 CRYPTO_BLAKE2S_QueryInstall()

Description

Query BLAKE2s hardware accelerator.

Prototype

```
void CRYPTO_BLAKE2S_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                  const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.3.4 Generic API

The following table lists the BLAKE2s functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_BLAKE2S_Init()</code>	Initialize context.
<code>CRYPTO_HASH_BLAKE2S_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_BLAKE2S_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_BLAKE2S_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_BLAKE2S_Kill()</code>	Destroy digest.

5.3.4.1 CRYPTO_HASH_BLAKE2S_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_BLAKE2S_Add(      void      * pContext,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.3.4.2 CRYPTO_HASH_BLAKE2S_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_BLAKE2S_Final(void * pContext,  
                                U8      * pDigest,  
                                unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.3.4.3 CRYPTO_HASH_BLAKE2S_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_BLAKE2S_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.3.4.4 CRYPTO_HASH_BLAKE2S_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_BLAKE2S_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.3.4.5 CRYPTO_HASH_BLAKE2S_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_BLAKE2S_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.3.5 Self-test API

The following table lists the BLAKE2s self-test API functions.

Function	Description
CRYPTO_BLAKE2S_RFC7693_SelfTest()	Run BLAKE2 KATs from RFC 7693.

5.3.5.1 CRYPTO_BLAKE2S RFC7693 SelfTest()

Description

Run BLAKE2 KATs from RFC 7693.

Prototype

```
void CRYPTO_BLAKE2S_RFC7693_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.4 MD5

5.4.1 Standards reference

MD5 is specified by the following document:

- IETF RFC 1321 — *The MD5 Message-Digest Algorithm*

5.4.2 Algorithm parameters

5.4.2.1 Block size

```
#define CRYPTO_MD5_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single MD5 block.

5.4.2.2 Digest size

```
#define CRYPTO_MD5_DIGEST_BIT_COUNT 128  
#define CRYPTO_MD5_DIGEST_BYTE_COUNT 16
```

The number of bits and bytes required to hold a complete MD5 digest.

```
#define CRYPTO_MD5_96_DIGEST_BYTE_COUNT (96/8)
```

The number of bytes required to hold a truncated MD5 digest of 96 bits.

5.4.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_MD5_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the MD5 hash functions for size rather than for speed. When optimized for speed, the MD5 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	Flash	0.3 KB	0.4 KB	0.7 KB
1	0.16 KB	-	-	2.0 KB	2.0 KB

5.4.4 Type-safe API

The following table lists the MD5 type-safe API functions.

Function	Description
Message functions	
CRYPTO_MD5_Calc()	Calculate digest.
CRYPTO_MD5_Calc_160()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_MD5_Init()	Initialize context.
CRYPTO_MD5_Add()	Add data to digest.
CRYPTO_MD5_Get()	Get incremental digest.
CRYPTO_MD5_Final()	Finalize digest calculation.
CRYPTO_MD5_Final_160()	Finalize digest calculation, fixed size.
CRYPTO_MD5_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_MD5_Install()	Install MD5 hash implementation.
CRYPTO_MD5_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_MD5_QueryInstall()	Query MD5 hardware accelerator.

5.4.4.1 CRYPTO_MD5_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_MD5_Add(      CRYPTO_MD5_CONTEXT * pSelf,
                           const U8          * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.4.4.2 CRYPTO_MD5_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_MD5_Calc(      U8      * pOutput,
                           unsigned   OutputLen,
                           const U8    * pInput,
                           unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.4.4.3 CRYPTO_MD5_Calc_160()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_MD5_Calc_160(      U8      * pOutput,
                               const U8      * pInput,
                               unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 20 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.4.4.4 CRYPTO_MD5_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_MD5_Final(CRYPTO_MD5_CONTEXT * pSelf,  
                      U8 * pOutput,  
                      unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.4.4.5 CRYPTO_MD5_Final_160()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_MD5_Final_160(CRYPTO_MD5_CONTEXT * pSelf,  
                           U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 20 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.4.4.6 CRYPTO_MD5_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_MD5_Get(CRYPTO_MD5_CONTEXT * pSelf,  
                      U8                 * pOutput,  
                      unsigned           OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.4.4.7 CRYPTO_MD5_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MD5_Init(CRYPTO_MD5_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to hash context.

5.4.4.8 CRYPTO_MD5_Install()

Description

Install MD5 hash implementation.

Prototype

```
void CRYPTO_MD5_Install(const CRYPTO_HASH_API * pHWAPI,  
                        const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.4.4.9 CRYPTO_MD5_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_MD5_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.4.4.10 CRYPTO_MD5_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_MD5_Kill(CRYPTO_MD5_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.4.4.11 CRYPTO_MD5_QueryInstall()

Description

Query MD5 hardware accelerator.

Prototype

```
void CRYPTO_MD5_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                           const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.4.5 Generic API

The following table lists the MD5 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_MD5_Init()</code>	Initialize context.
<code>CRYPTO_HASH_MD5_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_MD5_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_MD5_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_MD5_Kill()</code>	Destroy digest.

5.4.5.1 CRYPTO_HASH_MD5_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_MD5_Add(      void      * pContext,
                               const U8      * pInput,
                               unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.4.5.2 CRYPTO_HASH_MD5_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_MD5_Final(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.4.5.3 CRYPTO_HASH_MD5_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_MD5_Get(void * pContext,  
                         U8      * pDigest,  
                         unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.4.5.4 CRYPTO_HASH_MD5_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_MD5_Init(void * pContext);
```

Parameters

Parameter	Description
<code>pContext</code>	Pointer to hash context.

5.4.5.5 CRYPTO_HASH_MD5_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_MD5_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.4.6 Self-test API

The following table lists the MD5 self-test API functions.

Function	Description
CRYPTO_MD5_RFC1321_SelfTest()	Run MD5 test vectors from RFC 1321.

5.4.6.1 CRYPTO_MD5_RFC1321_SelfTest()

Description

Run MD5 test vectors from RFC 1321.

Prototype

```
void CRYPTO_MD5_RFC1321_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.4.7 Example applications

5.4.7.1 CRYPTO_Bench_MD5.c

This application benchmarks the configured performance of MD5. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
MD5 Benchmark compiled Mar 19 2018 16:34:02

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_CONFIG_MD5_OPTIMIZE = 1
Config: CRYPTO_CONFIG_MD5_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| MD5      |    25.10 |
+-----+-----+

Benchmark complete
```

Complete listing

```
*****
*           (c) SEGGER Microcontroller GmbH          *
*           The Embedded Experts                  *
*           www.segger.com                      *
*****  

----- END-OF-HEADER -----  

File      : CRYPTO_Bench_MD5.c
Purpose   : Benchmark MD5 implementation.  

*/  

*****  

*  

*      #include section  

*  

*****  

*/  

#include "CRYPTO.h"
#include "SEGGER_SYS.h"  

*****  

*  

*      Static data  

*  

*****  

*/  

static U8 _aTestMessage[65536] = { 0 };  

*****  

*  

*      Static code  

*  

*****  

*/  

*****  

*      _ConvertTicksToSeconds()  

*  

*  Function description  

*  Convert ticks to seconds.  

*  

*  Parameters  

*  Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
```

```

/*
 *  Return value
 *      Number of seconds corresponding to tick.
 */
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

//*****************************************************************************
/*
 *      _HashBenchmark()
 *
 *  Function description
 *      Benchmarks a hash implementation.
 *
 *  Parameters
 *      sAlgorithm - Hash algorithm name.
 *      pAPI       - Pointer to hash API.
 */
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_MD5_CONTEXT C;
    U64          T0;
    U64          OneSecond;
    unsigned     n;
    //
    SEGGER_SYS_IO_Printf("| %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfClaim) {
        pAPI->pfClaim();
    }
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

//*****************************************************************************
/*
 *      Public code
 */
//*****************************************************************************
/*
 *      MainTask()
 *
 *  Function description
 *      Main entry point for application to run all the tests.
 */
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API *pHWAPI;
    const CRYPTO_HASH_API *pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("MD5 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
        (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION           = %u\n",
    [__FILE__], CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_MD5_OPTIMIZE = %d\n", CRYPTO_CONFIG_MD5_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_MD5_HW_OPTIMIZE = %d\n",
    CRYPTO_CONFIG_MD5_HW_OPTIMIZE);
    //
    SEGGER_SYS_IO_Printf("+-----+-----+\n");
    SEGGER_SYS_IO_Printf("| Algorithm | Hash MB/s |\n");
    SEGGER_SYS_IO_Printf("+-----+-----+\n");
    //
    _HashBenchmark("MD5", &CRYPTO_HASH_MD5_SW);
    CRYPTO_MD5_QueryInstall(&pHWAPI, &pSWAPI);
    if (pHWAPI && pHWAPI != &CRYPTO_HASH_MD5_SW) {
        _HashBenchmark("MD5 (HW)", pHWAPI);
}

```

```
}

SEGGER_SYS_IO_Printf( "+-----+\n" );
//  
SEGGER_SYS_IO_Printf( "\nBenchmark complete\n" );
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****
```

5.5 RIPEMD-160

5.5.1 Standards reference

RIPEMD-160 is specified by the following document:

- Antoon Bosselaers — *The hash function RIPEMD-160*

5.5.2 Algorithm parameters

5.5.2.1 Block size

```
#define CRYPTO_RIPEMD160_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single RIPEMD-160 block.

5.5.2.2 Digest size

```
#define CRYPTO_RIPEMD160_DIGEST_BIT_COUNT 160  
#define CRYPTO_RIPEMD160_DIGEST_BYTE_COUNT 20
```

The number of bits and bytes required to hold a complete RIPEMD-160 digest.

5.5.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_RIPEMD160_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the RIPEMD-160 hash functions for size rather than for speed. When optimized for speed, the RIPEMD-160 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	Flash	0.3 KB	0.7 KB	1.0 KB
1	0.16 KB	-	-	4.6 KB	4.6 KB

5.5.4 Type-safe API

The following table lists the RIPEMD-160 type-safe API functions.

Function	Description
Message functions	
CRYPTO_RIPEMD160_Calc()	Calculate digest.
CRYPTO_RIPEMD160_Calc_160()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_RIPEMD160_Init()	Initialize context.
CRYPTO_RIPEMD160_Add()	Add data to digest.
CRYPTO_RIPEMD160_Get()	Get incremental digest.
CRYPTO_RIPEMD160_Final()	Finalize digest calculation.
CRYPTO_RIPEMD160_Final_160()	Finalize digest calculation, fixed size.
CRYPTO_RIPEMD160_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_RIPEMD160_Install()	Install RIPEMD-160 hash implementation.
CRYPTO_RIPEMD160_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_RIPEMD160_QueryInstall()	Query RIPEMD-160 hardware accelerator.

5.5.4.1 CRYPTO_RIPEMD160_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_RIPEMD160_Add(      CRYPTO_RIPEMD160_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.5.4.2 CRYPTO_RIPEMD160_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_RIPEMD160_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pInput,
      unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.5.4.3 CRYPTO_RIPEMD160_Calc_160()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_RIPEMD160_Calc_160(      U8      * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 20 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.5.4.4 CRYPTO_RIPEMD160_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_RIPEMD160_Final(CRYPTO_RIPEMD160_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.5.4.5 CRYPTO_RIPEMD160_Final_160()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_RIPEMD160_Final_160(CRYPTO_RIPEMD160_CONTEXT * pSelf,  
                                  U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 20 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.5.4.6 CRYPTO_RIPEMD160_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_RIPEMD160_Get(CRYPTO_RIPEMD160_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.5.4.7 CRYPTO_RIPEMD160_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_RIPEMD160_Init(CRYPTO_RIPEMD160_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.5.4.8 CRYPTO_RIPEMD160_Install()

Description

Install RIPEMD-160 hash implementation.

Prototype

```
void CRYPTO_RIPEMD160_Install(const CRYPTO_HASH_API * pHwapi,  
                               const CRYPTO_HASH_API * pSwapi);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.5.4.9 CRYPTO_RIPEMD160_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_RIPEMD160_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.5.4.10 CRYPTO_RIPEMD160_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_RIPEMD160_Kill(CRYPTO_RIPEMD160_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.5.4.11 CRYPTO_RIPEMD160_QueryInstall()

Description

Query RIPEMD-160 hardware accelerator.

Prototype

```
void CRYPTO_RIPEMD160_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                     const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.5.5 Generic API

The following table lists the RIPEMD-160 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_RIPEMD160_Init()</code>	Initialize context.
<code>CRYPTO_HASH_RIPEMD160_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_RIPEMD160_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_RIPEMD160_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_RIPEMD160_Kill()</code>	Destroy digest.

5.5.5.1 CRYPTO_HASH_RIPEMD160_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_RIPEMD160_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.5.5.2 CRYPTO_HASH_RIPEMD160_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_RIPEMD160_Final(void * pContext,  
                                U8      * pDigest,  
                                unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.5.5.3 CRYPTO_HASH_RIPEMD160_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_RIPEMD160_Get(void * pContext,  
                                U8      * pDigest,  
                                unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.5.5.4 CRYPTO_HASH_RIPEMD160_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_RIPEMD160_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.5.5.5 CRYPTO_HASH_RIPEMD160_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_RIPEMD160_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.5.6 Self-test API

The following table lists the RIPEMD-160 self-test API functions.

Function	Description
CRYPTO_RIPEMD160_Bosselaers_SelfTest()	Run all RIPEMD160 test vectors defined by Bosselaers.

5.5.6.1 CRYPTO_RIPEMD160_Bosselaers_SelfTest()

Description

Run all RIPEMD160 test vectors defined by Bosselaers.

Prototype

```
void CRYPTO_RIPEMD160_Bosselaers_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.5.7 Example applications

5.5.7.1 CRYPTO_Bench_RIPEMD160.c

This application benchmarks the configured performance of RIPEMD-160. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
RIPEMD160 Benchmark compiled Mar 19 2018 16:42:14

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed                  = 200.000 MHz
Config: CRYPTO_VERSION                 = 22400 [2.24]
Config: CRYPTO_CONFIG_RIPEMD160_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| RIPEMD160 (SW) |     8.47 |
+-----+-----+

Benchmark complete
```

Complete listing

```
/****************************************************************************
 *          (c) SEGGER Microcontroller GmbH           *
 *          The Embedded Experts                   *
 *          www.segger.com                      *
 ****
 ----- END-OF-HEADER -----
File      : CRYPTO_Bench_RIPEMD160.c
Purpose   : Benchmark RIPEMD-160 implementation.

*/
/*
*      #include section
*
*****
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
*      Static const data
*
*****
*/
static const U8 _aTestMessage[65536] = { 0 };

/*
*      Static code
*
*****
*/
_CovertTicksToSeconds()

/*
*      Function description
*      Convert ticks to seconds.
*
*      Parameters
*      Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
```

```

*   Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

*****_HashBenchmark()
*
*   Function description
*   Benchmarks a hash implementation.
*
*   Parameters
*   sAlgorithm - Hash algorithm name.
*   pAPI       - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA512_CONTEXT C; // big enough for most things...
    U64                  T0;
    U64                  OneSecond;
    unsigned             n;
    //
    SEGGER_SYS_IO_Printf("| %-14s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****_Public code
*****
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API *pHWAPI;
    const CRYPTO_HASH_API *pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("RIPEMD160 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
                             (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION          = %u\n"
                         "[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_RIPEMD160_OPTIMIZE = %d\n"
                         "\n", CRYPTO_CONFIG_RIPEMD160_OPTIMIZE);
    //
    SEGGER_SYS_IO_Printf("-----+-----+\n");
    SEGGER_SYS_IO_Printf(" | Algorithm      | Hash MB/s | \n");
    SEGGER_SYS_IO_Printf("-----+-----+\n");
    //
    _HashBenchmark("RIPEMD160 (SW)", &CRYPTO_HASH_RIPEMD160_SW);
    CRYPTO_RIPEMD160_QueryInstall(&pHWAPI, &pSWAPI);
    if (pHWAPI != &CRYPTO_HASH_RIPEMD160_SW) {
        _HashBenchmark("RIPEMD160 (HW)", pHWAPI);
    }
    SEGGER_SYS_IO_Printf("-----+-----+\n");
    SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
    SEGGER_SYS_OS_Halt(0);
}

```

```
}
```

```
***** End of file *****
```

5.6 SHA-1

5.6.1 Standards reference

SHA-1 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.6.2 Algorithm parameters

5.6.2.1 Block size

```
#define CRYPTO_SHA1_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single SHA-1 block.

5.6.2.2 Digest size

```
#define CRYPTO_SHA1_DIGEST_BIT_COUNT 160  
#define CRYPTO_SHA1_DIGEST_BYTE_COUNT 20
```

The number of bits and bytes required to hold a complete SHA-1 digest.

```
#define CRYPTO_SHA1_96_DIGEST_BYTE_COUNT (96/8)
```

The number of bytes required to hold a truncated SHA-1 digest of 96 bits.

5.6.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_SHA1_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the SHA-1 hash functions for size rather than for speed. When optimized for speed, the SHA-1 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M4 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	-	-	0.6 KB	0.6 KB
1	0.16 KB	-	-	3.6 KB	3.6 KB

5.6.4 Type-safe API

The following table lists the SHA-1 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA1_Calc()	Calculate digest.
CRYPTO_SHA1_Calc_160()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA1_Init()	Initialize context.
CRYPTO_SHA1_Add()	Add data to digest.
CRYPTO_SHA1_Get()	Get incremental digest.
CRYPTO_SHA1_Final()	Finalize digest calculation.
CRYPTO_SHA1_Final_160()	Finalize digest calculation, fixed size.
CRYPTO_SHA1_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA1_Install()	Install SHA-1 hash implementation.
CRYPTO_SHA1_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA1_QueryInstall()	Query SHA-1 hardware accelerator.

5.6.4.1 CRYPTO_SHA1_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA1_Add(      CRYPTO_SHA1_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.6.4.2 CRYPTO_SHA1_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA1_Calc(      U8      * pOutput,
                           unsigned OutputLen,
                           const U8     * pInput,
                           unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.6.4.3 CRYPTO_SHA1_Calc_160()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA1_Calc_160(      U8      * pOutput,
                                const U8      * pInput,
                                unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 20 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.6.4.4 CRYPTO_SHA1_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA1_Final(CRYPTO_SHA1_CONTEXT * pSelf,  
                        U8                 * pOutput,  
                        unsigned           OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.6.4.5 CRYPTO_SHA1_Final_160()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA1_Final_160(CRYPTO_SHA1_CONTEXT * pSelf,  
                           U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 20 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.6.4.6 CRYPTO_SHA1_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA1_Get(CRYPTO_SHA1_CONTEXT * pSelf,  
                      U8                 * pOutput,  
                      unsigned           OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.6.4.7 CRYPTO_SHA1_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA1_Init(CRYPTO_SHA1_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.6.4.8 CRYPTO_SHA1_Install()

Description

Install SHA-1 hash implementation.

Prototype

```
void CRYPTO_SHA1_Install(const CRYPTO_HASH_API * pHWAPI,  
                         const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.6.4.9 CRYPTO_SHA1_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA1_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.6.4.10 CRYPTO_SHA1_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA1_Kill(CRYPTO_SHA1_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.6.4.11 CRYPTO_SHA1_QueryInstall()

Description

Query SHA-1 hardware accelerator.

Prototype

```
void CRYPTO_SHA1_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                           const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.6.5 Generic API

The following table lists the SHA-1 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA1_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA1_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA1_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA1_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA1_Kill()</code>	Destroy digest.

5.6.5.1 CRYPTO_HASH_SHA1_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA1_Add(      void      * pContext,
                                const U8      * pInput,
                                unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.6.5.2 CRYPTO_HASH_SHA1_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA1_Final(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.6.5.3 CRYPTO_HASH_SHA1_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA1_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.6.5.4 CRYPTO_HASH_SHA1_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA1_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.6.5.5 CRYPTO_HASH_SHA1_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA1_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.6.6 Self-test API

The following table lists the SHA-1 self-test API functions.

Function	Description
<code>CRYPTO_SHA1_FIPS180_SelfTest()</code>	Run SHA-1 KATs from FIPS 180-2.
<code>CRYPTO_SHA1_CAVS_SelfTest()</code>	Run SHA-1 KATs from CAVS.

5.6.6.1 CRYPTO_SHA1_CAVS_SelfTest()

Description

Run SHA-1 KATs from CAVS.

Prototype

```
void CRYPTO_SHA1_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.6.6.2 CRYPTO_SHA1_FIPS180_SelfTest()

Description

Run SHA-1 KATs from FIPS 180-2.

Prototype

```
void CRYPTO_SHA1_FIPS180_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.6.7 Example applications

5.6.7.1 CRYPTO_Bench_SHA1.c

This application benchmarks the configured performance of SHA-1. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
SHA-1 Benchmark compiled Mar 19 2018 16:42:46

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA1_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA1_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-1     |    11.68 |
| SHA-1 (HW) |    65.51 |
+-----+-----+

Benchmark complete
```

Complete listing

```
*****
*           (c) SEGGER Microcontroller GmbH          *
*           The Embedded Experts                  *
*           www.segger.com                      *
*****  

----- END-OF-HEADER -----  

File      : CRYPTO_Bench_SHA1.c
Purpose   : Benchmark SHA-1 implementation.  

*/  

*****  

*  

*      #include section  

*  

*****  

*/  

#include "CRYPTO.h"
#include "SEGGER_SYS.h"  

*****  

*  

*      Static const data  

*  

*****  

*/  

static const U8 _aTestMessage[65536] = { 0 };  

*****  

*  

*      Static code  

*  

*****  

*/  

*****  

*  

*      _ConvertTicksToSeconds()  

*  

*  Function description
*  Convert ticks to seconds.
*
```

```

* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

//****************************************************************************
*
*     _HashBenchmark()
*
* Function description
*   Benchmarks a hash implementation.
*
* Parameters
*   sAlgorithm - Hash algorithm name.
*   pAPI      - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA512_CONTEXT C; // big enough for most things...
    U64                  T0;
    U64                  OneSecond;
    unsigned             n;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfClaim) {
        pAPI->pfClaim();
    }
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

//****************************************************************************
*
*     Public code
*
*****
*/
*****
```

```

*****
```

```

*     MainTask()
*
* Function description
*   Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API * pHWAPI;
    const CRYPTO_HASH_API * pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("SHA-1 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
                            (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION          = %u\n",
    [%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA1_OPTIMIZE = %d\n",
    CRYPTO_CONFIG_SHA1_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA1_HW_OPTIMIZE = %d\n\n",
    CRYPTO_CONFIG_SHA1_HW_OPTIMIZE);
    //
    SEGGER_SYS_IO_Printf("-----+\n");
    SEGGER_SYS_IO_Printf(" | Algorithm   | Hash MB/s | \n");
    SEGGER_SYS_IO_Printf("-----+\n");
    //
    _HashBenchmark("SHA-1", &CRYPTO_HASH_SHA1_SW);
}
```

```
CRYPTO_SHA1_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI && pHWAPI != &CRYPTO_HASH_SHA1_SW) {
    _HashBenchmark("SHA-1 (HW)", pHWAPI);
}
SEGGER_SYS_IO_Printf("+-----+-----+\n");
//SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****
```

5.7 SHA-224

5.7.1 Standards reference

SHA-224 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.7.2 Algorithm parameters

5.7.2.1 Block size

```
#define CRYPTO_SHA224_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single SHA-224 block.

5.7.2.2 Digest size

```
#define CRYPTO_SHA224_DIGEST_BIT_COUNT 224  
#define CRYPTO_SHA224_DIGEST_BYTE_COUNT 28
```

The number of bit and bytes required to hold a complete SHA-1 digest.

5.7.3 Type-safe API

The following table lists the SHA-224 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_SHA224_Calc()</code>	Calculate digest.
<code>CRYPTO_SHA224_Calc_224()</code>	Calculate digest, fixed size.
Incremental functions	
<code>CRYPTO_SHA224_Init()</code>	Initialize context.
<code>CRYPTO_SHA224_Add()</code>	Add data to digest.
<code>CRYPTO_SHA224_Get()</code>	Get incremental digest.
<code>CRYPTO_SHA224_Final()</code>	Finalize digest calculation.
<code>CRYPTO_SHA224_Final_224()</code>	Finalize digest calculation, fixed size.
<code>CRYPTO_SHA224_Kill()</code>	Destroy context.
Setup and hardware acceleration	
<code>CRYPTO_SHA224_Install()</code>	Install SHA-224 hash implementation.
<code>CRYPTO_SHA224_IsInstalled()</code>	Query whether hash algorithm is installed.
<code>CRYPTO_SHA224_QueryInstall()</code>	Query SHA-224 hardware accelerator.

5.7.3.1 CRYPTO_SHA224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA224_Add(          CRYPTO_SHA224_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.7.3.2 CRYPTO_SHA224_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA224_Calc(      U8      * pOutput,
                               unsigned   OutputLen,
                           const U8      * pInput,
                               unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.7.3.3 CRYPTO_SHA224_Calc_224()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA224_Calc_224(      U8      * pOutput,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 28 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.7.3.4 CRYPTO_SHA224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA224_Final(CRYPTO_SHA224_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.7.3.5 CRYPTO_SHA224_Final_224()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA224_Final_224(CRYPTO_SHA224_CONTEXT * pSelf,  
                               U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 28 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.7.3.6 CRYPTO_SHA224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA224_Get(CRYPTO_SHA224_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.7.3.7 CRYPTO_SHA224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA224_Init(CRYPTO_SHA224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.7.3.8 CRYPTO_SHA224_Install()

Description

Install SHA-224 hash implementation.

Prototype

```
void CRYPTO_SHA224_Install(const CRYPTO_HASH_API * pHwAPI,  
                           const CRYPTO_HASH_API * pSwAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.7.3.9 CRYPTO_SHA224_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA224_IsInstalled(void);
```

Return value

- = 0 Hash algorithm is not installed.
- ≠ 0 Hash algorithm is installed.

5.7.3.10 CRYPTO_SHA224_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA224_Kill(CRYPTO_SHA224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.7.3.11 CRYPTO_SHA224_QueryInstall()

Description

Query SHA-224 hardware accelerator.

Prototype

```
void CRYPTO_SHA224_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.7.4 Generic API

The following table lists the SHA-224 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA224_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA224_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA224_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA224_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA224_Kill()</code>	Destroy digest.

5.7.4.1 CRYPTO_HASH_SHA224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA224_Add(      void      * pContext,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.7.4.2 CRYPTO_HASH_SHA224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA224_Final(void      * pContext,
                               U8        * pDigest,
                               unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.7.4.3 CRYPTO_HASH_SHA224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA224_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.7.4.4 CRYPTO_HASH_SHA224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA224_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.7.4.5 CRYPTO_HASH_SHA224_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.7.5 Self-test API

The following table lists the SHA-224 self-test API functions.

Function	Description
CRYPTO_SHA224_CAVS_SelfTest()	Run SHA-224 KATs from CAVS.

5.7.5.1 CRYPTO_SHA224_CAVS_SelfTest()

Description

Run SHA-224 KATs from CAVS.

Prototype

```
void CRYPTO_SHA224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.8 SHA-256

5.8.1 Standards reference

SHA-256 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.8.2 Algorithm parameters

5.8.2.1 Block size

```
#define CRYPTO_SHA256_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single SHA-256 block.

5.8.2.2 Digest size

```
#define CRYPTO_SHA256_DIGEST_BIT_COUNT 256  
#define CRYPTO_SHA256_DIGEST_BYTE_COUNT 32
```

The number of bits and bytes required to hold a complete SHA-256 digest.

5.8.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_SHA256_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the SHA-256 hash functions for size rather than for speed. When optimized for speed, the SHA-256 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.17 KB	Flash	0.3 KB	0.5 KB	0.8 KB
1	0.17 KB	-	-	7.7 KB	7.7 KB

5.8.4 Type-safe API

The following table lists the SHA-256 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA256_Calc()	Calculate digest.
CRYPTO_SHA256_Calc_256()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA256_Init()	Initialize context.
CRYPTO_SHA256_Add()	Add data to digest.
CRYPTO_SHA256_Get()	Get incremental digest.
CRYPTO_SHA256_Final()	Finalize digest calculation.
CRYPTO_SHA256_Final_256()	Finalize digest calculation, fixed size.
CRYPTO_SHA256_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA256_Install()	Install SHA-256 hash implementation.
CRYPTO_SHA256_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA256_QueryInstall()	Query SHA-256 hardware accelerator.

5.8.4.1 CRYPTO_SHA256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA256_Add(          CRYPTO_SHA256_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.8.4.2 CRYPTO_SHA256_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA256_Calc(      U8      * pOutput,
                               unsigned   OutputLen,
                           const U8      * pInput,
                               unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.8.4.3 CRYPTO_SHA256_Calc_256()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA256_Calc_256(      U8      * pOutput,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 32 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.8.4.4 CRYPTO_SHA256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA256_Final(CRYPTO_SHA256_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.8.4.5 CRYPTO_SHA256_Final_256()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA256_Final_256(CRYPTO_SHA256_CONTEXT * pSelf,  
                               U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.8.4.6 CRYPTO_SHA256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA256_Get(CRYPTO_SHA256_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.8.4.7 CRYPTO_SHA256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA256_Init(CRYPTO_SHA256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.8.4.8 CRYPTO_SHA256_Install()

Description

Install SHA-256 hash implementation.

Prototype

```
void CRYPTO_SHA256_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.8.4.9 CRYPTO_SHA256_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA256_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.8.4.10 CRYPTO_SHA256_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA256_Kill(CRYPTO_SHA256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.8.4.11 CRYPTO_SHA256_QueryInstall()

Description

Query SHA-256 hardware accelerator.

Prototype

```
void CRYPTO_SHA256_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.8.5 Generic API

The following table lists the SHA-256 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA256_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA256_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA256_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA256_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA256_Kill()</code>	Destroy digest.

5.8.5.1 CRYPTO_HASH_SHA256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA256_Add(      void      * pContext,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.8.5.2 CRYPTO_HASH_SHA256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA256_Final(void      * pContext,
                               U8        * pDigest,
                               unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.8.5.3 CRYPTO_HASH_SHA256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA256_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.8.5.4 CRYPTO_HASH_SHA256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA256_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.8.5.5 CRYPTO_HASH_SHA256_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.8.6 Self-test API

The following table lists the SHA-256 self-test API functions.

Function	Description
CRYPTO_SHA256_FIPS180_SelfTest()	Run SHA-256 KATs from FIPS 180-2.
CRYPTO_SHA256_CAVS_SelfTest()	Run SHA-256 KATs from CAVS.

5.8.6.1 CRYPTO_SHA256_CAVS_SelfTest()

Description

Run SHA-256 KATs from CAVS.

Prototype

```
void CRYPTO_SHA256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.8.6.2 CRYPTO_SHA256_FIPS180_SelfTest()

Description

Run SHA-256 KATs from FIPS 180-2.

Prototype

```
void CRYPTO_SHA256_FIPS180_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.8.7 Example applications

5.8.7.1 CRYPTO_Bench_SHA256.c

This application benchmarks the configured performance of SHA-256. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
SHA-256 Benchmark compiled Mar 19 2018 16:23:21

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed                  = 200.000 MHz
Config: CRYPTO_VERSION                 = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA256_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA256_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-256 (SW) |      3.61 |
| SHA-256 (HW) |    112.94 |
+-----+-----+

Benchmark complete
```

Complete listing

```
***** (c) SEGGER Microcontroller GmbH *****
*           The Embedded Experts          *
*           www.segger.com               *
***** END-OF-HEADER *****

File      : CRYPTO_Bench_SHA256.c
Purpose   : Benchmark SHA-256 implementation.

*/
***** #include section *****
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
*       Static data
*
***** Static code *****
*/
static const U8 _aTestMessage[8192] = { 0 };

/*
*       _ConvertTicksToSeconds()
*
* Function description
*   Convert ticks to seconds.
*/
```

```

* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

*****_HashBenchmark()

* Function description
*   Benchmarks a hash implementation.
*
* Parameters
*   sAlgorithm - Hash algorithm name.
*   pAPI      - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA256_CONTEXT C;
    U64                  T0;
    U64                  OneSecond;
    unsigned             n;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfClaim) {
        pAPI->pfClaim();
    }
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****Public code
*****
* MainTask()
*
* Function description
*   Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API * pHWAPI;
    const CRYPTO_HASH_API * pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s     www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("SHA-256 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
        "(double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f");
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION           = %u\n",
    "[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA256_OPTIMIZE = %d\n",
    "\n", CRYPTO_CONFIG_SHA256_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA256_HW_OPTIMIZE = %d\n",
    "\n", CRYPTO_CONFIG_SHA256_HW_OPTIMIZE);
    //
    SEGGER_SYS_IO_Printf("-----+\n");
    SEGGER_SYS_IO_Printf(" | Algorithm   | Hash MB/s | \n");
    SEGGER_SYS_IO_Printf("-----+\n");
    //
    _HashBenchmark("SHA-224 (SW)", &CRYPTO_HASH_SHA224_SW);
}

```

```
CRYPTO_SHA224_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI && pHWAPI != &CRYPTO_HASH_SHA224_SW) {
    _HashBenchmark("SHA-224 (HW)", pHWAPI);
}
_HashBenchmark("SHA-256 (SW)", &CRYPTO_HASH_SHA256_SW);
CRYPTO_SHA256_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI && pHWAPI != &CRYPTO_HASH_SHA256_SW) {
    _HashBenchmark("SHA-256 (HW)", pHWAPI);
}
SEGGER_SYS_IO_Printf("+-----+\n");
//SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****
```

5.9 SHA-384

5.9.1 Standards reference

SHA-384 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.9.2 Algorithm parameters

5.9.2.1 Block size

```
#define CRYPTO_SHA384_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single SHA-384 block.

5.9.2.2 Digest size

```
#define CRYPTO_SHA384_DIGEST_BIT_COUNT 384  
#define CRYPTO_SHA384_DIGEST_BYTE_COUNT 48
```

The number of bits and bytes required to hold a complete SHA-384 digest.

5.9.3 Type-safe API

The following table lists the SHA-384 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA384_Calc()	All-in-one computation of SHA-384 digest over data.
CRYPTO_SHA384_Calc_384()	Calculate digest over message.
Incremental functions	
CRYPTO_SHA384_Init()	Initialize context.
CRYPTO_SHA384_Add()	Add data to digest.
CRYPTO_SHA384_Get()	Get incremental digest.
CRYPTO_SHA384_Final()	Finalize digest calculation.
CRYPTO_SHA384_Final_384()	Finalize digest calculation, fixed size.
CRYPTO_SHA384_Kill()	Destroy context.

5.9.3.1 CRYPTO_SHA384_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA384_Add(      CRYPTO_SHA384_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.9.3.2 CRYPTO_SHA384_Calc()

Description

All-in-one computation of SHA-384 digest over data.

Prototype

```
void CRYPTO_SHA384_Calc(      U8      * pOutput,
                               unsigned   OutputLen,
                           const U8      * pInput,
                               unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.9.3.3 CRYPTO_SHA384_Calc_384()

Description

Calculate digest over message.

Prototype

```
void CRYPTO_SHA384_Calc_384(      U8      * pOutput,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 48 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.9.3.4 CRYPTO_SHA384_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA384_Final(CRYPTO_SHA384_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.9.3.5 CRYPTO_SHA384_Final_384()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA384_Final_384(CRYPTO_SHA384_CONTEXT * pSelf,  
                               U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.9.3.6 CRYPTO_SHA384_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA384_Get(CRYPTO_SHA384_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.9.3.7 CRYPTO_SHA384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA384_Init(CRYPTO_SHA384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.9.3.8 CRYPTO_SHA384_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA384_Kill(CRYPTO_SHA384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.9.4 Generic API

The following table lists the SHA-384 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA384_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA384_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA384_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA384_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA384_Kill()</code>	Destroy digest.

5.9.4.1 CRYPTO_HASH_SHA384_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA384_Add(      void      * pContext,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.9.4.2 CRYPTO_HASH_SHA384_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA384_Final(void      * pContext,
                               U8        * pDigest,
                               unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.9.4.3 CRYPTO_HASH_SHA384_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA384_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.9.4.4 CRYPTO_HASH_SHA384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA384_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.9.4.5 CRYPTO_HASH_SHA384_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA384_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.9.5 Self-test API

The following table lists the SHA-384 self-test API functions.

Function	Description
CRYPTO_SHA384_CAVS_SelfTest()	Run SHA-384 KATs from CAVS.

5.9.5.1 CRYPTO_SHA384_CAVS_SelfTest()

Description

Run SHA-384 KATs from CAVS.

Prototype

```
void CRYPTO_SHA384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.10 SHA-512

5.10.1 Standards reference

SHA-512 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.10.2 Algorithm parameters

5.10.2.1 Block size

```
#define CRYPTO_SHA512_BLOCK_BYTE_COUNT 128
```

The number of bytes in a single SHA-512 block.

5.10.2.2 Digest size

```
#define CRYPTO_SHA512_DIGEST_BIT_COUNT 512
#define CRYPTO_SHA512_DIGEST_BYTE_COUNT 64
```

The number of bits and bytes required to hold a complete SHA-512 digest.

5.10.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_SHA512_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the SHA-512 hash functions for size rather than for speed. When optimized for speed, the SHA-512 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.20 KB	Flash	0.7 KB	1.1 KB	1.8 KB
1	0.20 KB	Flash	0.7 KB	10.3 KB	11.0 KB
2	0.20 KB	Flash	0.1 KB	41.5 KB	41.6 KB

5.10.4 Type-safe API

The following table lists the SHA-512 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_SHA512_Calc()</code>	Calculate digest.
<code>CRYPTO_SHA512_Calc_512()</code>	Calculate digest, fixed size.
Incremental functions	
<code>CRYPTO_SHA512_Init()</code>	Initialize context.
<code>CRYPTO_SHA512_Add()</code>	Add data to digest.
<code>CRYPTO_SHA512_Get()</code>	Get incremental digest.
<code>CRYPTO_SHA512_Final()</code>	Finalize digest calculation.
<code>CRYPTO_SHA512_Final_512()</code>	Finalize digest calculation, fixed size.
<code>CRYPTO_SHA512_Kill()</code>	Destroy context.
Setup and hardware acceleration	
<code>CRYPTO_SHA512_Install()</code>	Install SHA-512 hash implementation.
<code>CRYPTO_SHA512_IsInstalled()</code>	Query whether hash algorithm is installed.
<code>CRYPTO_SHA512_QueryInstall()</code>	Query SHA-512 hardware accelerator.

5.10.4.1 CRYPTO_SHA512_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA512_Add(      CRYPTO_SHA512_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.10.4.2 CRYPTO_SHA512_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA512_Calc(      U8      * pOutput,
                               unsigned   OutputLen,
                           const U8      * pInput,
                               unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.10.4.3 CRYPTO_SHA512_Calc_512()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA512_Calc_512(      U8      * pOutput,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 64 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.10.4.4 CRYPTO_SHA512_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA512_Final(CRYPTO_SHA512_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.10.4.5 CRYPTO_SHA512_Final_512()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA512_Final_512(CRYPTO_SHA512_CONTEXT * pSelf,  
                               U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 64 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.10.4.6 CRYPTO_SHA512_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA512_Get(CRYPTO_SHA512_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.10.4.7 CRYPTO_SHA512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA512_Init(CRYPTO_SHA512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.10.4.8 CRYPTO_SHA512_Install()

Description

Install SHA-512 hash implementation.

Prototype

```
void CRYPTO_SHA512_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.10.4.9 CRYPTO_SHA512_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA512_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.10.4.10 CRYPTO_SHA512_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA512_Kill(CRYPTO_SHA512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.10.4.11 CRYPTO_SHA512_QueryInstall()

Description

Query SHA-512 hardware accelerator.

Prototype

```
void CRYPTO_SHA512_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                 const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.10.5 Generic API

The following table lists the SHA-512 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA512_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA512_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA512_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA512_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA512_Kill()</code>	Destroy digest.

5.10.5.1 CRYPTO_HASH_SHA512_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA512_Add(      void      * pContext,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.10.5.2 CRYPTO_HASH_SHA512_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA512_Final(void      * pContext,
                               U8        * pDigest,
                               unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.10.5.3 CRYPTO_HASH_SHA512_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA512_Get(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.10.5.4 CRYPTO_HASH_SHA512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA512_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.10.5.5 CRYPTO_HASH_SHA512_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA512_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.10.6 Self-test API

The following table lists the SHA-512 self-test API functions.

Function	Description
<code>CRYPTO_SHA512_FIPS180_SelfTest()</code>	Run SHA-512 KATs from FIPS 180-2.
<code>CRYPTO_SHA512_CAVS_SelfTest()</code>	Run SHA-512 KATs from CAVS.

5.10.6.1 CRYPTO_SHA512_CAVS_SelfTest()

Description

Run SHA-512 KATs from CAVS.

Prototype

```
void CRYPTO_SHA512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.10.6.2 CRYPTO_SHA512_FIPS180_SelfTest()

Description

Run SHA-512 KATs from FIPS 180-2.

Prototype

```
void CRYPTO_SHA512_FIPS180_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.10.7 Example applications

5.10.7.1 CRYPTO_Bench_SHA512.c

This application benchmarks the configured performance of SHA-512. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
SHA-512 Benchmark compiled Mar 19 2018 16:43:06

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed                  = 200.000 MHz
Config: CRYPTO_VERSION                 = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA512_OPTIMIZE = 2
Config: CRYPTO_CONFIG_SHA512_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-512 (SW) |      1.57 |
+-----+-----+

Benchmark complete
```

Complete listing

```
*****
*          (c) SEGGER Microcontroller GmbH      *
*          The Embedded Experts              *
*          www.segger.com                  *
*****
----- END-OF-HEADER -----

File      : CRYPTO_Bench_SHA512.c
Purpose   : Benchmark SHA-512 implementation.

*/
***** #include section *****
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

*****
*      Static const data
*
***** */
static const U8 _aTestMessage[65536] = { 0 };

*****
*      Static code
*
***** */
(ConvertTicksToSeconds())
*
Function description
Convert ticks to seconds.
*
Parameters
Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
```

```

/*
 *  Return value
 *      Number of seconds corresponding to tick.
 */
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

//*****************************************************************************
/*
*      _HashBenchmark()
*
*  Function description
*      Benchmarks a hash implementation.
*
*  Parameters
*      sAlgorithm - Hash algorithm name.
*      pAPI       - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA512_CONTEXT C; // big enough for most things...
    U64                  T0;
    U64                  OneSecond;
    unsigned             n;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

//*****************************************************************************
/*
*      Public code
*
*****
*/
*****
```

```

/*
*      MainTask()
*
*  Function description
*      Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API * pHWAPI;
    const CRYPTO_HASH_API * pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("SHA-512 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
                            (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION           = %u\n",
                         [s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA512_OPTIMIZE = %d\n",
                         \n", CRYPTO_CONFIG_SHA512_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_SHA512_HW_OPTIMIZE = %d\n",
                         \n", CRYPTO_CONFIG_SHA256_HW_OPTIMIZE);
    //
    SEGGER_SYS_IO_Printf("-----+-----+\n");
    SEGGER_SYS_IO_Printf(" | Algorithm   | Hash MB/s | \n");
    SEGGER_SYS_IO_Printf("-----+-----+\n");
    //
    _HashBenchmark("SHA-512 (SW)", &CRYPTO_HASH_SHA512_SW);
    CRYPTO_SHA512_QueryInstall(&pHWAPI, &pSWAPI);
    if (pHWAPI != &CRYPTO_HASH_SHA512_SW) {
        _HashBenchmark("SHA-512 (HW)", pHWAPI);
    }
    SEGGER_SYS_IO_Printf("-----+-----+\n");
```

```
//  
SEGGER_SYS_IO_Printf("\nBenchmark complete\n");  
SEGGER_SYS_OS_PauseBeforeHalt();  
SEGGER_SYS_OS_Halt(0);  
}  
  
***** End of file *****
```

5.11 SHA-512/224

5.11.1 Standards reference

SHA-512/224 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.11.2 Algorithm parameters

5.11.2.1 Block size

```
#define CRYPTO_SHA512_224_BLOCK_BYTE_COUNT 128
```

The number of bytes in a single SHA-512/224 block.

5.11.2.2 Digest size

```
#define CRYPTO_SHA512_224_DIGEST_BIT_COUNT 224  
#define CRYPTO_SHA512_224_DIGEST_BYTE_COUNT 28
```

The number of bits and bytes required to hold a complete SHA-512/224 digest.

5.11.3 Type-safe API

The following table lists the SHA-512/224 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_SHA512_224_Calc()</code>	Calculate digest.
<code>CRYPTO_SHA512_224_Calc_224()</code>	Calculate digest, fixed size.
Incremental functions	
<code>CRYPTO_SHA512_224_Init()</code>	Initialize context.
<code>CRYPTO_SHA512_224_Add()</code>	Add data to digest.
<code>CRYPTO_SHA512_224_Get()</code>	Get incremental digest.
<code>CRYPTO_SHA512_224_Final()</code>	Finalize digest calculation.
<code>CRYPTO_SHA512_224_Final_224()</code>	Finish digest calculation, fixed size.
<code>CRYPTO_SHA512_224_Kill()</code>	Destroy context.

5.11.3.1 CRYPTO_SHA512_224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA512_224_Add(          CRYPTO_SHA512_224_CONTEXT * pSelf,
                                    const U8           * pInput,
                                    unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.11.3.2 CRYPTO_SHA512_224_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA512_224_Calc(      U8      * pOutput,
                                    unsigned OutputLen,
                                    const U8     * pInput,
                                    unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

5.11.3.3 CRYPTO_SHA512_224_Calc_224()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA512_224_Calc_224(      U8      * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 28 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

5.11.3.4 CRYPTO_SHA512_224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA512_224_Final(CRYPTO_SHA512_224_CONTEXT * pSelf,  
                               U8 * pOutput,  
                               unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.11.3.5 CRYPTO_SHA512_224_Final_224()

Description

Finish digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA512_224_Final_224(CRYPTO_SHA512_224_CONTEXT * pSelf,  
                                  U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 28 bytes.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.11.3.6 CRYPTO_SHA512_224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA512_224_Get(CRYPTO_SHA512_224_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives message digest.
OutputLen	Octet length of the digest.

Additional information

This function calculates the intermediate SHA-512/256 digest from the data that has been added. After calling this function, the context is not destroyed and additional data can be added to continue digest calculation.

5.11.3.7 CRYPTO_SHA512_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA512_224_Init(CRYPTO_SHA512_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.11.3.8 CRYPTO_SHA512_224_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA512_224_Kill(CRYPTO_SHA512_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.11.4 Generic API

The following table lists the SHA-512/224 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA512_224_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA512_224_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA512_224_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA512_224_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA512_224_Kill()</code>	Destroy digest.

5.11.4.1 CRYPTO_HASH_SHA512_224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA512_224_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.11.4.2 CRYPTO_HASH_SHA512_224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA512_224_Final(void      * pContext,
                                    U8        * pDigest,
                                    unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.11.4.3 CRYPTO_HASH_SHA512_224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA512_224_Get(void      * pContext,
                                  U8        * pDigest,
                                  unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.11.4.4 CRYPTO_HASH_SHA512_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA512_224_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.11.4.5 CRYPTO_HASH_SHA512_224_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA512_224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.12 SHA-512/256

5.12.1 Standards reference

SHA-512/256 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

5.12.2 Algorithm parameters

5.12.2.1 Block size

```
#define CRYPTO_SHA512_256_BLOCK_BYTE_COUNT 128
```

The number of bytes in a single SHA-512/256 block.

5.12.2.2 Digest size

```
#define CRYPTO_SHA512_256_DIGEST_BIT_COUNT 256  
#define CRYPTO_SHA512_256_DIGEST_BYTE_COUNT 32
```

The number of bits and bytes required to hold a complete SHA-512/256 digest.

5.12.3 Type-safe API

The following table lists the SHA-512/256 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_SHA512_256_Calc()</code>	Calculate digest over message.
<code>CRYPTO_SHA512_256_Calc_256()</code>	Calculate digest over message.
Incremental functions	
<code>CRYPTO_SHA512_256_Init()</code>	Initialize context.
<code>CRYPTO_SHA512_256_Add()</code>	Add data to digest.
<code>CRYPTO_SHA512_256_Get()</code>	Get incremental digest.
<code>CRYPTO_SHA512_256_Final()</code>	Finalize digest calculation.
<code>CRYPTO_SHA512_256_Final_256()</code>	Finish digest calculation, fixed size.
<code>CRYPTO_SHA512_256_Kill()</code>	Destroy context.

5.12.3.1 CRYPTO_SHA512_256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA512_256_Add(          CRYPTO_SHA512_256_CONTEXT * pSelf,
                                    const U8           * pInput,
                                    unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.12.3.2 CRYPTO_SHA512_256_Calc()

Description

Calculate digest over message.

Prototype

```
void CRYPTO_SHA512_256_Calc(      U8      * pOutput,
                                    unsigned OutputLen,
                                    const U8     * pInput,
                                    unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.
pInput	Pointer to input octet string to hash.
InputLen	Octet length of the input octet string.

5.12.3.3 CRYPTO_SHA512_256_Calc_256()

Description

Calculate digest over message.

Prototype

```
void CRYPTO_SHA512_256_Calc_256(      U8      * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest.
pInput	Pointer to input octet string to hash.
InputLen	Octet length of the input octet string.

5.12.3.4 CRYPTO_SHA512_256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA512_256_Final(CRYPTO_SHA512_256_CONTEXT * pSelf,  
                               U8 * pOutput,  
                               unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.12.3.5 CRYPTO_SHA512_256_Final_256()

Description

Finish digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA512_256_Final_256(CRYPTO_SHA512_256_CONTEXT * pSelf,  
                                  U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.12.3.6 CRYPTO_SHA512_256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA512_256_Get(CRYPTO_SHA512_256_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

5.12.3.7 CRYPTO_SHA512_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA512_256_Init(CRYPTO_SHA512_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.12.3.8 CRYPTO_SHA512_256_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA512_256_Kill(CRYPTO_SHA512_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.12.4 Generic API

The following table lists the SHA-512/256 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA512_256_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA512_256_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA512_256_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA512_256_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA512_256_Kill()</code>	Destroy digest.

5.12.4.1 CRYPTO_HASH_SHA512_256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA512_256_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.12.4.2 CRYPTO_HASH_SHA512_256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA512_256_Final(void * pContext,  
                                  U8 * pDigest,  
                                  unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.12.4.3 CRYPTO_HASH_SHA512_256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA512_256_Get(void      * pContext,
                                  U8        * pDigest,
                                  unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.12.4.4 CRYPTO_HASH_SHA512_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA512_256_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.12.4.5 CRYPTO_HASH_SHA512_256_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA512_256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13 SHA3-224

5.13.1 Standards reference

SHA3-224 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

5.13.2 Algorithm parameters

5.13.2.1 Block size

```
#define CRYPTO_SHA3_224_BLOCK_BYTE_COUNT 144
```

The number of bytes in a single SHA3-224 block.

5.13.2.2 Digest size

```
#define CRYPTO_SHA3_224_DIGEST_BIT_COUNT 224  
#define CRYPTO_SHA3_224_DIGEST_BYTE_COUNT 28
```

The number of bit and bytes required to hold a complete SHA3-1 digest.

5.13.3 Type-safe API

The following table lists the SHA3-224 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA3_224_Calc()	Calculate digest.
CRYPTO_SHA3_224_Calc_224()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA3_224_Init()	Initialize context.
CRYPTO_SHA3_224_Add()	Add data to digest.
CRYPTO_SHA3_224_Get()	Get incremental digest.
CRYPTO_SHA3_224_Final()	Finalize digest calculation.
CRYPTO_SHA3_224_Final_224()	Finalize digest calculation, fixed size.
CRYPTO_SHA3_224_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA3_224_Install()	Install SHA3-224 hash implementation.
CRYPTO_SHA3_224_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA3_224_QueryInstall()	Query SHA3-224 hardware accelerator.

5.13.3.1 CRYPTO_SHA3_224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA3_224_Add(      CRYPTO_SHA3_224_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.13.3.2 CRYPTO_SHA3_224_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA3_224_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8    * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.13.3.3 CRYPTO_SHA3_224_Calc_224()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA3_224_Calc_224(      U8      * pOutput,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 28 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.13.3.4 CRYPTO_SHA3_224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA3_224_Final(CRYPTO_SHA3_224_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13.3.5 CRYPTO_SHA3_224_Final_224()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA3_224_Final_224(CRYPTO_SHA3_224_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 28 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13.3.6 CRYPTO_SHA3_224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA3_224_Get(CRYPTO_SHA3_224_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.13.3.7 CRYPTO_SHA3_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA3_224_Init(CRYPTO_SHA3_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.13.3.8 CRYPTO_SHA3_224_Install()

Description

Install SHA3-224 hash implementation.

Prototype

```
void CRYPTO_SHA3_224_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.13.3.9 CRYPTO_SHA3_224_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA3_224_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.13.3.10 CRYPTO_SHA3_224_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA3_224_Kill(CRYPTO_SHA3_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13.3.11 CRYPTO_SHA3_224_QueryInstall()

Description

Query SHA3-224 hardware accelerator.

Prototype

```
void CRYPTO_SHA3_224_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                   const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.13.4 Generic API

The following table lists the SHA3-224 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA3_224_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA3_224_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA3_224_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA3_224_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA3_224_Kill()</code>	Destroy digest.

5.13.4.1 CRYPTO_HASH_SHA3_224_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA3_224_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.13.4.2 CRYPTO_HASH_SHA3_224_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA3_224_Final(void      * pContext,
                                U8        * pDigest,
                                unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13.4.3 CRYPTO_HASH_SHA3_224_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA3_224_Get(void * pContext,  
                               U8 * pDigest,  
                               unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.13.4.4 CRYPTO_HASH_SHA3_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA3_224_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.13.4.5 CRYPTO_HASH_SHA3_224_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA3_224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.13.5 Self-test API

The following table lists the SHA3-224 self-test API functions.

Function	Description
<code>CRYPTO_SHA3_224_FIPS202_SelfTest()</code>	Run SHA3-224 KATs from FIPS 202.
<code>CRYPTO_SHA3_224_CAVS_SelfTest()</code>	Run SHA3-224 KATs from CAVS.

5.13.5.1 CRYPTO_SHA3_224_CAVS_SelfTest()

Description

Run SHA3-224 KATs from CAVS.

Prototype

```
void CRYPTO_SHA3_224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.13.5.2 CRYPTO_SHA3_224_FIPS202_SelfTest()

Description

Run SHA3-224 KATs from FIPS 202.

Prototype

```
void CRYPTO_SHA3_224_FIPS202_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.14 SHA3-256

5.14.1 Standards reference

SHA3-256 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

5.14.2 Algorithm parameters

5.14.2.1 Block size

```
#define CRYPTO_SHA3_256_BLOCK_BYTE_COUNT 136
```

The number of bytes in a single SHA3-256 block.

5.14.2.2 Digest size

```
#define CRYPTO_SHA3_256_DIGEST_BIT_COUNT 256  
#define CRYPTO_SHA3_256_DIGEST_BYTE_COUNT 32
```

The number of bits and bytes required to hold a complete SHA3-256 digest.

5.14.3 Type-safe API

The following table lists the SHA3-256 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA3_256_Calc()	Calculate digest.
CRYPTO_SHA3_256_Calc_256()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA3_256_Init()	Initialize context.
CRYPTO_SHA3_256_Add()	Add data to digest.
CRYPTO_SHA3_256_Get()	Get incremental digest.
CRYPTO_SHA3_256_Final()	Finalize digest calculation.
CRYPTO_SHA3_256_Final_256()	Finalize digest calculation, fixed size.
CRYPTO_SHA3_256_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA3_256_Install()	Install SHA3-256 hash implementation.
CRYPTO_SHA3_256_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA3_256_QueryInstall()	Query SHA3-256 hardware accelerator.

5.14.3.1 CRYPTO_SHA3_256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA3_256_Add(      CRYPTO_SHA3_256_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.14.3.2 CRYPTO_SHA3_256_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA3_256_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8    * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.14.3.3 CRYPTO_SHA3_256_Calc_256()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA3_256_Calc_256(      U8      * pOutput,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 32 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.14.3.4 CRYPTO_SHA3_256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA3_256_Final(CRYPTO_SHA3_256_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.14.3.5 CRYPTO_SHA3_256_Final_256()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA3_256_Final_256(CRYPTO_SHA3_256_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.14.3.6 CRYPTO_SHA3_256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA3_256_Get(CRYPTO_SHA3_256_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.14.3.7 CRYPTO_SHA3_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA3_256_Init(CRYPTO_SHA3_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.14.3.8 CRYPTO_SHA3_256_Install()

Description

Install SHA3-256 hash implementation.

Prototype

```
void CRYPTO_SHA3_256_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.14.3.9 CRYPTO_SHA3_256_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA3_256_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.14.3.10 CRYPTO_SHA3_256_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA3_256_Kill(CRYPTO_SHA3_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.14.3.11 CRYPTO_SHA3_256_QueryInstall()

Description

Query SHA3-256 hardware accelerator.

Prototype

```
void CRYPTO_SHA3_256_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                   const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.14.4 Generic API

The following table lists the SHA3-256 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA3_256_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA3_256_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA3_256_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA3_256_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA3_256_Kill()</code>	Destroy digest.

5.14.4.1 CRYPTO_HASH_SHA3_256_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA3_256_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.14.4.2 CRYPTO_HASH_SHA3_256_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA3_256_Final(void      * pContext,
                                U8        * pDigest,
                                unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.14.4.3 CRYPTO_HASH_SHA3_256_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA3_256_Get(void * pContext,  
                               U8 * pDigest,  
                               unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.14.4.4 CRYPTO_HASH_SHA3_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA3_256_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.14.4.5 CRYPTO_HASH_SHA3_256_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA3_256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.14.5 Self-test API

The following table lists the SHA3-256 self-test API functions.

Function	Description
<code>CRYPTO_SHA3_256_FIPS202_SelfTest()</code>	Run SHA3-256 KATs from FIPS 202.
<code>CRYPTO_SHA3_256_CAVS_SelfTest()</code>	Run SHA3-256 KATs from CAVS.

5.14.5.1 CRYPTO_SHA3_256_CAVS_SelfTest()

Description

Run SHA3-256 KATs from CAVS.

Prototype

```
void CRYPTO_SHA3_256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.14.5.2 CRYPTO_SHA3_256_FIPS202_SelfTest()

Description

Run SHA3-256 KATs from FIPS 202.

Prototype

```
void CRYPTO_SHA3_256_FIPS202_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.15 SHA3-384

5.15.1 Standards reference

SHA3-384 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

5.15.2 Algorithm parameters

5.15.2.1 Block size

```
#define CRYPTO_SHA3_384_BLOCK_BYTE_COUNT 104
```

The number of bytes in a single SHA3-384 block.

5.15.2.2 Digest size

```
#define CRYPTO_SHA3_384_DIGEST_BIT_COUNT 384  
#define CRYPTO_SHA3_384_DIGEST_BYTE_COUNT 48
```

The number of bits and bytes required to hold a complete SHA3-384 digest.

5.15.3 Type-safe API

The following table lists the SHA3-384 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA3_384_Calc()	Calculate digest.
CRYPTO_SHA3_384_Calc_384()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA3_384_Init()	Initialize context.
CRYPTO_SHA3_384_Add()	Add data to digest.
CRYPTO_SHA3_384_Get()	Get incremental digest.
CRYPTO_SHA3_384_Final()	Finalize digest calculation.
CRYPTO_SHA3_384_Final_384()	Finalize digest calculation, fixed size.
CRYPTO_SHA3_384_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA3_384_Install()	Install SHA3-384 hash implementation.
CRYPTO_SHA3_384_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA3_384_QueryInstall()	Query SHA3-384 hardware accelerator.

5.15.3.1 CRYPTO_SHA3_384_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA3_384_Add(      CRYPTO_SHA3_384_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.15.3.2 CRYPTO_SHA3_384_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA3_384_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8     * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.15.3.3 CRYPTO_SHA3_384_Calc_384()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA3_384_Calc_384(      U8      * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 48 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.15.3.4 CRYPTO_SHA3_384_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA3_384_Final(CRYPTO_SHA3_384_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.15.3.5 CRYPTO_SHA3_384_Final_384()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA3_384_Final_384(CRYPTO_SHA3_384_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 48 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.15.3.6 CRYPTO_SHA3_384_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA3_384_Get(CRYPTO_SHA3_384_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.15.3.7 CRYPTO_SHA3_384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA3_384_Init(CRYPTO_SHA3_384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.15.3.8 CRYPTO_SHA3_384_Install()

Description

Install SHA3-384 hash implementation.

Prototype

```
void CRYPTO_SHA3_384_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.15.3.9 CRYPTO_SHA3_384_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA3_384_IsInstalled(void);
```

Return value

= 0 Hash algorithm is not installed.
≠ 0 Hash algorithm is installed.

5.15.3.10 CRYPTO_SHA3_384_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA3_384_Kill(CRYPTO_SHA3_384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.15.3.11 CRYPTO_SHA3_384_QueryInstall()

Description

Query SHA3-384 hardware accelerator.

Prototype

```
void CRYPTO_SHA3_384_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                   const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.15.4 Generic API

The following table lists the SHA3-384 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA3_384_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA3_384_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA3_384_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA3_384_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA3_384_Kill()</code>	Destroy digest.

5.15.4.1 CRYPTO_HASH_SHA3_384_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA3_384_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.15.4.2 CRYPTO_HASH_SHA3_384_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA3_384_Final(void      * pContext,
                                U8        * pDigest,
                                unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.15.4.3 CRYPTO_HASH_SHA3_384_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA3_384_Get(void      * pContext,
                               U8        * pDigest,
                               unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.15.4.4 CRYPTO_HASH_SHA3_384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA3_384_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.15.4.5 CRYPTO_HASH_SHA3_384_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA3_384_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.15.5 Self-test API

The following table lists the SHA3-384 self-test API functions.

Function	Description
<code>CRYPTO_SHA3_384_FIPS202_SelfTest()</code>	Run SHA3-384 KATs from FIPS 202.
<code>CRYPTO_SHA3_384_CAVS_SelfTest()</code>	Run SHA3-384 KATs from CAVS.

5.15.5.1 CRYPTO_SHA3_384_CAVS_SelfTest()

Description

Run SHA3-384 KATs from CAVS.

Prototype

```
void CRYPTO_SHA3_384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.15.5.2 CRYPTO_SHA3_384_FIPS202_SelfTest()

Description

Run SHA3-384 KATs from FIPS 202.

Prototype

```
void CRYPTO_SHA3_384_FIPS202_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.16 SHA3-512

5.16.1 Standards reference

SHA3-512 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

5.16.2 Algorithm parameters

5.16.2.1 Block size

```
#define CRYPTO_SHA3_512_BLOCK_BYTE_COUNT 72
```

The number of bytes in a single SHA3-512 block.

5.16.2.2 Digest size

```
#define CRYPTO_SHA3_512_DIGEST_BIT_COUNT 512
#define CRYPTO_SHA3_512_DIGEST_BYTE_COUNT 64
```

The number of bits and bytes required to hold a complete SHA3-512 digest.

5.16.3 Type-safe API

The following table lists the SHA3-512 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SHA3_512_Calc()	Calculate digest.
CRYPTO_SHA3_512_Calc_512()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SHA3_512_Init()	Initialize context.
CRYPTO_SHA3_512_Add()	Add data to digest.
CRYPTO_SHA3_512_Get()	Get incremental digest.
CRYPTO_SHA3_512_Final()	Finalize digest calculation.
CRYPTO_SHA3_512_Final_512()	Finalize digest calculation, fixed size.
CRYPTO_SHA3_512_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SHA3_512_Install()	Install SHA3-512 hash implementation.
CRYPTO_SHA3_512_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SHA3_512_QueryInstall()	Query SHA3-512 hardware accelerator.

5.16.3.1 CRYPTO_SHA3_512_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SHA3_512_Add(      CRYPTO_SHA3_512_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.16.3.2 CRYPTO_SHA3_512_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SHA3_512_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8    * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.16.3.3 CRYPTO_SHA3_512_Calc_512()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SHA3_512_Calc_512(      U8      * pOutput,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 64 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.16.3.4 CRYPTO_SHA3_512_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SHA3_512_Final(CRYPTO_SHA3_512_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.16.3.5 CRYPTO_SHA3_512_Final_512()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SHA3_512_Final_512(CRYPTO_SHA3_512_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 64 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.16.3.6 CRYPTO_SHA3_512_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SHA3_512_Get(CRYPTO_SHA3_512_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.16.3.7 CRYPTO_SHA3_512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SHA3_512_Init(CRYPTO_SHA3_512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.16.3.8 CRYPTO_SHA3_512_Install()

Description

Install SHA3-512 hash implementation.

Prototype

```
void CRYPTO_SHA3_512_Install(const CRYPTO_HASH_API * pHWAPI,  
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.16.3.9 CRYPTO_SHA3_512_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SHA3_512_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.16.3.10 CRYPTO_SHA3_512_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SHA3_512_Kill(CRYPTO_SHA3_512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.16.3.11 CRYPTO_SHA3_512_QueryInstall()

Description

Query SHA3-512 hardware accelerator.

Prototype

```
void CRYPTO_SHA3_512_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                                   const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.16.4 Generic API

The following table lists the SHA3-512 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SHA3_512_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SHA3_512_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SHA3_512_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SHA3_512_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SHA3_512_Kill()</code>	Destroy digest.

5.16.4.1 CRYPTO_HASH_SHA3_512_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SHA3_512_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.16.4.2 CRYPTO_HASH_SHA3_512_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SHA3_512_Final(void      * pContext,
                                U8        * pDigest,
                                unsigned   DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.16.4.3 CRYPTO_HASH_SHA3_512_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SHA3_512_Get(void * pContext,  
                               U8      * pDigest,  
                               unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.16.4.4 CRYPTO_HASH_SHA3_512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SHA3_512_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.16.4.5 CRYPTO_HASH_SHA3_512_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SHA3_512_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.16.5 Self-test API

The following table lists the SHA3-512 self-test API functions.

Function	Description
<code>CRYPTO_SHA3_512_FIPS202_SelfTest()</code>	Run SHA3-512 KATs from FIPS 202.
<code>CRYPTO_SHA3_512_CAVS_SelfTest()</code>	Run SHA3-512 KATs from CAVS.

5.16.5.1 CRYPTO_SHA3_512_CAVS_SelfTest()

Description

Run SHA3-512 KATs from CAVS.

Prototype

```
void CRYPTO_SHA3_512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.16.5.2 CRYPTO_SHA3_512_FIPS202_SelfTest()

Description

Run SHA3-512 KATs from FIPS 202.

Prototype

```
void CRYPTO_SHA3_512_FIPS202_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.17 SM3

5.17.1 Standards reference

SM3 is specified by the following document:

- [draft-sca-cfrg-sm3-02 — The SM3 Cryptographic Hash Function](#)

5.17.2 Algorithm parameters

5.17.2.1 Block size

```
#define CRYPTO_SM3_BLOCK_BYTE_COUNT 64
```

The number of bytes in a single SM3 block.

5.17.2.2 Digest size

```
#define CRYPTO_SM3_DIGEST_BIT_COUNT 256  
#define CRYPTO_SM3_DIGEST_BYTE_COUNT 32
```

The number of bits and bytes required to hold a complete SM3 digest.

5.17.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_SM3_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol to zero to optimize the SM3 hash functions for size rather than for speed. When optimized for speed, the SM3 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.17 KB	Flash	0.3 KB	0.7 KB	1.0 KB
1	0.17 KB	-	-	8.2 KB	8.2 KB

5.17.4 Type-safe API

The following table lists the SM3 type-safe API functions.

Function	Description
Message functions	
CRYPTO_SM3_Calc()	Calculate digest.
CRYPTO_SM3_Calc_256()	Calculate digest, fixed size.
Incremental functions	
CRYPTO_SM3_Init()	Initialize context.
CRYPTO_SM3_Add()	Add data to digest.
CRYPTO_SM3_Get()	Get incremental digest.
CRYPTO_SM3_Final()	Finalize digest calculation.
CRYPTO_SM3_Final_256()	Finalize digest calculation, fixed size.
CRYPTO_SM3_Kill()	Destroy context.
Setup and hardware acceleration	
CRYPTO_SM3_Install()	Install SM3 hash implementation.
CRYPTO_SM3_IsInstalled()	Query whether hash algorithm is installed.
CRYPTO_SM3_QueryInstall()	Query SM3 hardware accelerator.

5.17.4.1 CRYPTO_SM3_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_SM3_Add(      CRYPTO_SM3_CONTEXT * pSelf,
                           const U8          * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

5.17.4.2 CRYPTO_SM3_Calc()

Description

Calculate digest.

Prototype

```
void CRYPTO_SM3_Calc(      U8      * pOutput,
                           unsigned   OutputLen,
                           const U8    * pInput,
                           unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the message digest.
<code>OutputLen</code>	Octet length of the message digest.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.17.4.3 CRYPTO_SM3_Calc_256()

Description

Calculate digest, fixed size.

Prototype

```
void CRYPTO_SM3_Calc_256(      U8      * pOutput,
                               const U8      * pInput,
                               unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 32 octets.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

Additional information

It is possible to truncate the digest by specifying OutputLen less than the full digest length: in this case, the leftmost (most significant) octets of the digest are written to the message digest buffer.

5.17.4.4 CRYPTO_SM3_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_SM3_Final(CRYPTO_SM3_CONTEXT * pSelf,  
                      U8 * pOutput,  
                      unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.17.4.5 CRYPTO_SM3_Final_256()

Description

Finalize digest calculation, fixed size.

Prototype

```
void CRYPTO_SM3_Final_256(CRYPTO_SM3_CONTEXT * pSelf,  
                           U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest, 32 octets.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.17.4.6 CRYPTO_SM3_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_SM3_Get(CRYPTO_SM3_CONTEXT * pSelf,  
                      U8                 * pOutput,  
                      unsigned           OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.17.4.7 CRYPTO_SM3_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_SM3_Init(CRYPTO_SM3_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

5.17.4.8 CRYPTO_SM3_Install()

Description

Install SM3 hash implementation.

Prototype

```
void CRYPTO_SM3_Install(const CRYPTO_HASH_API * pHWAPI,  
                        const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

5.17.4.9 CRYPTO_SM3_IsInstalled()

Description

Query whether hash algorithm is installed.

Prototype

```
int CRYPTO_SM3_IsInstalled(void);
```

Return value

= 0	Hash algorithm is not installed.
≠ 0	Hash algorithm is installed.

5.17.4.10 CRYPTO_SM3_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_SM3_Kill(CRYPTO_SM3_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.17.4.11 CRYPTO_SM3_QueryInstall()

Description

Query SM3 hardware accelerator.

Prototype

```
void CRYPTO_SM3_QueryInstall(const CRYPTO_HASH_API ** ppHWAPI,  
                           const CRYPTO_HASH_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the preferred API pointer.
ppSWAPI	Pointer to object that receives the fallback API pointer.

5.17.5 Generic API

The following table lists the SM3 functions that conform to the generic hash API.

Function	Description
<code>CRYPTO_HASH_SM3_Init()</code>	Initialize context.
<code>CRYPTO_HASH_SM3_Add()</code>	Add data to digest.
<code>CRYPTO_HASH_SM3_Get()</code>	Get incremental digest.
<code>CRYPTO_HASH_SM3_Final()</code>	Finalize digest calculation.
<code>CRYPTO_HASH_SM3_Kill()</code>	Destroy digest.

5.17.5.1 CRYPTO_HASH_SM3_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_HASH_SM3_Add(      void      * pContext,
                               const U8      * pInput,
                               unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pInput	Pointer to octet string to add to digest.
InputLen	Octet length of the octet string.

5.17.5.2 CRYPTO_HASH_SM3_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_HASH_SM3_Final(void * pContext,  
                           U8      * pDigest,  
                           unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.17.5.3 CRYPTO_HASH_SM3_Get()

Description

Get incremental digest.

Prototype

```
void CRYPTO_HASH_SM3_Get(void * pContext,  
                         U8      * pDigest,  
                         unsigned DigestLen);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.
pDigest	Pointer to object that receives the message digest.
DigestLen	Octet length of the digest.

Additional information

This function computes the current message digest and writes it to the receiving object. The hash context is not invalidated and additional data can be added to the hash context in order to continue hashing.

5.17.5.4 CRYPTO_HASH_SM3_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HASH_SM3_Init(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

5.17.5.5 CRYPTO_HASH_SM3_Kill()

Description

Destroy digest.

Prototype

```
void CRYPTO_HASH_SM3_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to hash context.

Additional information

After calling this function, the context is destroyed and must be reinitialized to be used again. The entire hash context is set to zero to ensure no cryptographic material remains in memory.

5.17.6 Self-test API

The following table lists the SM3 self-test API functions.

Function	Description
CRYPTO_SM3_GBT_SelfTest()	Run SM3 KATs from GBT.

5.17.6.1 CRYPTO_SM3_GBT_SelfTest()

Description

Run SM3 KATs from GBT.

Prototype

```
void CRYPTO_SM3_GBT_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

5.18 GHASH

5.18.1 Type-safe API

The following table lists the GHASH type-safe API functions.

Function	Description
Message functions	
CRYPTO_GHASH_Calc()	Calculate digest over message.
Incremental functions	
CRYPTO_GHASH_InitEx()	Initialize context.
CRYPTO_GHASH_Add()	Add data to digest.
CRYPTO_GHASH_Final()	Finalize digest calculation.
CRYPTO_GHASH_Kill()	Destroy context.

5.18.1.1 CRYPTO_GHASH_Add()

Description

Add data to digest.

Prototype

```
void CRYPTO_GHASH_Add(      CRYPTO_GHASH_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pInput	Pointer to input string to add.
InputLen	Octet length of the input string.

5.18.1.2 CRYPTO_GHASH_Calc()

Description

Calculate digest over message.

Prototype

```
void CRYPTO_GHASH_Calc(      U8      * pOutput,
                           const U8      * pSubkey,
                           const U8      * pInput,
                           unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the message digest, 16 octets.
pSubkey	Pointer to hash subkey, 16 octets.
pInput	Pointer to message to hash.
InputLen	Octet length of message.

5.18.1.3 CRYPTO_GHASH_Final()

Description

Finalize digest calculation.

Prototype

```
void CRYPTO_GHASH_Final(CRYPTO_GHASH_CONTEXT * pSelf,  
                         U8                      * pOutput,  
                         unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pOutput	Pointer to object that receives the message digest.
OutputLen	Octet length of the message digest.

5.18.1.4 CRYPTO_GHASH_InitEx()

Description

Initialize context.

Prototype

```
void CRYPTO_GHASH_InitEx(      CRYPTO_GHASH_CONTEXT * pSelf,
                               const U8             * pIV,
                               unsigned            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

5.18.1.5 CRYPTO_GHASH_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_GHASH_Kill(CRYPTO_GHASH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to hash context.

Chapter 6

MAC algorithms

emCrypt implements the following message authentication code algorithms:

- CMAC-AES
- CMAC-TDES
- CMAC-CAST
- CMAC-SEED
- CMAC-ARIA
- CMAC-Camellia
- CMAC-Twofish
- CMAC-Blowfish
- GMAC-AES
- GMAC-SEED
- GMAC-ARIA
- GMAC-Camellia
- GMAC-Twofish
- HMAC-MD5
- HMAC-RIPEMD-160
- HMAC-SHA-1
- HMAC-SHA-224
- HMAC-SHA-256
- HMAC-SHA-384
- HMAC-SHA-512
- HMAC-SHA-512/224
- HMAC-SHA-512/256
- HMAC-SHA3-224
- HMAC-SHA3-256
- HMAC-SHA3-384
- HMAC-SHA3-512
- XCBC-AES
- XCBC-SEED
- XCBC-ARIA
- XCBC-Camellia
- XCBC-Twofish
- KMAC
- Poly1305
- Poly1305-AES
- Poly1305-ARIA
- Poly1305-SEED
- Poly1305-Camellia

- Poly1305-Twofish
- Michael

6.1 Introduction

In general a MAC calculation is performed in three steps:

- Initialising the calculation using the key.
- Processing input data. This step can be repeated multiple times.
- Calculating the final MAC value.

The key and the intermediate results are stored in a data structure called a ‘MAC context’. The MAC context is maintained by the MAC functions, only the memory must be provided by the caller. It can be discarded after the final MAC calculation is done.

The API functions are named in the same way for all MAC algorithms:

- CRYPTO_<mac_algo_name>_Init() for initializing and setting the key.
- CRYPTO_<mac_algo_name>_Add() to process data.
- CRYPTO_<mac_algo_name>_Final() to calculate the final MAC value.

Example

```
//  
// Example for a SHA-1 HMAC calculation.  
//  
static const U8          Key[] = { 0x08, 0x15, 0x85, 0xa1, ..., 0x5b, 0xa3 };  
CRYPTO_HMAC_SHA1_CONTEXT HMACContext;  
U8                      aMAC[CRYPTO_SHA1_DIGEST_BYTE_COUNT];  
//  
// Initialize the hash context.  
//  
CRYPTO_HMAC_SHA1_Init(&HMACContext, Key, sizeof(Key));  
//  
// Process input data.  
//  
CRYPTO_HMAC_SHA1_Add(&HMACContext, Data1, Data1Len);  
//  
// More data.  
//  
CRYPTO_HMAC_SHA1_Add(&HMACContext, Data2, Data2Len);  
//  
// Calculate MAC.  
//  
CRYPTO_HMAC_SHA1_Final(&HMACContext, aMAC, sizeof(aMAC));  
//  
// aMAC now contains the MAC value.  
// From now, HMACContext is not used any more.  
//
```

For every MAC algorithm there is also a function to perform the whole MAC calculation in one step. These functions are called CRYPTO_<mac_algo_name>_Calc() and provide an easy way to calculate a MAC from a single piece of data.

Besides the type-safe API functions described above, there are also generic API functions, that use a void pointer to take the MAC context. These are useful, if the API functions shall be called via functions pointers to dynamically choose different MAC algorithms. When using the generic functions the caller is responsible to provide the correct context (or memory areas) via the void pointer argument.

6.2 CMAC-AES

6.2.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

AES is specified by the following document:

- FIPS PUB 197 — *Specification for the Advanced Encryption Standard (AES)*

6.2.2 Type-safe API

The following table lists the CMAC-AES type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_AES_Calc()	Calculate MAC.
CRYPTO_CMAC_AES_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_AES_Init()	Initialize context.
CRYPTO_CMAC_AES_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_AES_Add()	Add data to MAC.
CRYPTO_CMAC_AES_Final()	Finish MAC calculation.
CRYPTO_CMAC_AES_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_AES_Kill()	Destroy context.

6.2.2.1 CRYPTO_CMAC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_AES_Add(      CRYPTO_CMAC_AES_CONTEXT * pSelf,
                                const U8             * pInput,
                                unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.2.2.2 CRYPTO_CMAC_AES_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_AES_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC, 16 octets.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to cipher key.
<code>KeyLen</code>	Octet length of the cipher key.
<code>pInput</code>	Pointer to message.
<code>InputLen</code>	Octet length of the message.

6.2.2.3 CRYPTO_CMAC_AES_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_AES_Calc_128(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.2.2.4 CRYPTO_CMAC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_AES_Final(CRYPTO_CMAC_AES_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.2.2.5 CRYPTO_CMAC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_AES_Final_128(CRYPTO_CMAC_AES_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.2.2.6 CRYPTO_CMAC_AES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_AES_Init(      CRYPTO_CMAC_AES_CONTEXT * pSelf,
                                const U8             * pKey,
                                unsigned            KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.2.2.7 CRYPTO_CMAC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_AES_InitEx(          CRYPTO_CMAC_AES_CONTEXT * pSelf ,  
                                const U8                      * pKey ,  
                                unsigned                     KeyLen ,  
                                const U8                      * pIV ,  
                                unsigned                     IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.2.2.8 CRYPTO_CMAC_AES_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_AES_Kill(CRYPTO_CMAC_AES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.2.3 Generic API

The following table lists the CMAC-AES functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_AES_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_AES_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_AES_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_AES_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_AES_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_AES_Kill()</code>	Destroy MAC context.

6.2.3.1 CRYPTO_MAC_CMAC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_AES_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.2.3.2 CRYPTO_MAC_CMAC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_AES_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.2.3.3 CRYPTO_MAC_CMAC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_AES_Final_128(void * pContext,  
                                     U8     * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.2.3.4 CRYPTO_MAC_CMAC_AES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_AES_Init(      void      * pContext,
                                     const U8      * pKey,
                                     unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.2.3.5 CRYPTO_MAC_CMAC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_AES_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.2.3.6 CRYPTO_MAC_CMAC_AES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_AES_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.2.4 Self-test API

The following table lists the CMAC-AES self-test API functions.

Function	Description
CRYPTO_CMAC_AES_CAVS_SelfTest()	Run AES-CMAC self-test.

6.2.4.1 CRYPTO_CMAC_AES_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_CMAC_AES_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.3 CMAC-TDES

6.3.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

DES and TDES are specified by the following document:

- FIPS PUB 46-3 — *Data Encryption Standard (DES)*

6.3.2 Type-safe API

The following table lists the CMAC-TDES type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_TDES_Calc()	Calculate MAC.
CRYPTO_CMAC_TDES_Calc_64()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_TDES_Init()	Initialize context.
CRYPTO_CMAC_TDES_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_TDES_Add()	Add data to MAC.
CRYPTO_CMAC_TDES_Final()	Finish MAC calculation.
CRYPTO_CMAC_TDES_Final_64()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_TDES_Kill()	Destroy context.

6.3.2.1 CRYPTO_CMAC_TDES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_TDES_Add(      CRYPTO_CMAC_TDES_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.3.2.2 CRYPTO_CMAC_TDES_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_TDES_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned   KeyLen,
const U8      * pInput,
unsigned  InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.3.2.3 CRYPTO_CMAC_TDES_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_TDES_Calc_64(      U8      * pOutput,
                                       const U8      * pKey,
                                       unsigned      KeyLen,
                                       const U8      * pInput,
                                       unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.3.2.4 CRYPTO_CMAC_TDES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_TDES_Final(CRYPTO_CMAC_TDES_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.3.2.5 CRYPTO_CMAC_TDES_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_TDES_Final_64(CRYPTO_CMAC_TDES_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 8 octets.

6.3.2.6 CRYPTO_CMAC_TDES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_TDES_Init(      CRYPTO_CMAC_TDES_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.3.2.7 CRYPTO_CMAC_TDES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_TDES_InitEx(      CRYPTO_CMAC_TDES_CONTEXT * pSelf,
                                     const U8                  * pKey,
                                     unsigned                 KeyLen,
                                     const U8                  * pIV,
                                     unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.3.2.8 CRYPTO_CMAC_TDES_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_TDES_Kill(CRYPTO_CMAC_TDES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.3.3 Generic API

The following table lists the CMAC-TDES functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_TDES_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_TDES_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_TDES_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_TDES_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_TDES_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_TDES_Kill()</code>	Destroy MAC context.

6.3.3.1 CRYPTO_MAC_CMAC_TDES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.3.3.2 CRYPTO_MAC_CMAC_TDES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_Final(void      * pContext,
                                  U8        * pMAC,
                                  unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.3.3.3 CRYPTO_MAC_CMAC_TDES_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_Final_64(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.3.3.4 CRYPTO_MAC_CMAC_TDES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.3.3.5 CRYPTO_MAC_CMAC_TDES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.3.3.6 CRYPTO_MAC_CMAC_TDES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_TDES_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.3.4 Self-test API

The following table lists the CMAC-TDES self-test API functions.

Function	Description
CRYPTO_CMAC_TDES_CAVS_SelfTest()	Run AES-CMAC self-test.

6.3.4.1 CRYPTO_CMAC_TDES_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_CMAC_TDES_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.4 CMAC-IDEA

6.4.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

6.4.2 Type-safe API

The following table lists the CMAC-IDEA type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_IDEA_Calc()	Calculate MAC.
CRYPTO_CMAC_IDEA_Calc_64()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_IDEA_Init()	Initialize context.
CRYPTO_CMAC_IDEA_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_IDEA_Add()	Add data to MAC.
CRYPTO_CMAC_IDEA_Final()	Finish MAC calculation.
CRYPTO_CMAC_IDEA_Final_64()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_IDEA_Kill()	Destroy context.

6.4.2.1 CRYPTO_CMAC_IDEA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_IDEA_Add(          CRYPTO_CMAC_IDEA_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.4.2.2 CRYPTO_CMAC_IDEA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_IDEA_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned   KeyLen,
const U8      * pInput,
unsigned  InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC, 8 octets.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to cipher key.
<code>KeyLen</code>	Octet length of the cipher key.
<code>pInput</code>	Pointer to message.
<code>InputLen</code>	Octet length of the message.

6.4.2.3 CRYPTO_CMAC_IDEA_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_IDEA_Calc_64(      U8      * pOutput,
                                       const U8      * pKey,
                                       unsigned      KeyLen,
                                       const U8      * pInput,
                                       unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.4.2.4 CRYPTO_CMAC_IDEA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_IDEA_Final(CRYPTO_CMAC_IDEA_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.4.2.5 CRYPTO_CMAC_IDEA_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_IDEA_Final_64(CRYPTO_CMAC_IDEA_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 8 octets.

6.4.2.6 CRYPTO_CMAC_IDEA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_IDEA_Init(      CRYPTO_CMAC_IDEA_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.4.2.7 CRYPTO_CMAC_IDEA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_IDEA_InitEx(      CRYPTO_CMAC_IDEA_CONTEXT * pSelf,
                                     const U8                  * pKey,
                                     unsigned                 KeyLen,
                                     const U8                  * pIV,
                                     unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.4.2.8 CRYPTO_CMAC_IDEA_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_IDEA_Kill(CRYPTO_CMAC_IDEA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.4.3 Generic API

The following table lists the CMAC-IDEA functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_IDEA_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_IDEA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_IDEA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_IDEA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_IDEA_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_IDEA_Kill()</code>	Destroy MAC context.

6.4.3.1 CRYPTO_MAC_CMAC_IDEA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.4.3.2 CRYPTO_MAC_CMAC_IDEA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.4.3.3 CRYPTO_MAC_CMAC_IDEA_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_Final_64(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.4.3.4 CRYPTO_MAC_CMAC_IDEA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.4.3.5 CRYPTO_MAC_CMAC_IDEA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.4.3.6 CRYPTO_MAC_CMAC_IDEA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_IDEA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.5 CMAC-CAST

6.5.1 Type-safe API

The following table lists the CMAC-CAST type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_CAST_Calc()	Calculate MAC.
CRYPTO_CMAC_CAST_Calc_64()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_CAST_Init()	Initialize context.
CRYPTO_CMAC_CAST_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_CAST_Add()	Add data to MAC.
CRYPTO_CMAC_CAST_Final()	Finish MAC calculation.
CRYPTO_CMAC_CAST_Final_64()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_CAST_Kill()	Destroy context.

6.5.1.1 CRYPTO_CMAC_CAST_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_CAST_Add(          CRYPTO_CMAC_CAST_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.5.1.2 CRYPTO_CMAC_CAST_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_CAST_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
                                  unsigned KeyLen,
const U8      * pInput,
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.5.1.3 CRYPTO_CMAC_CAST_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_CAST_Calc_64(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.5.1.4 CRYPTO_CMAC_CAST_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_CAST_Final(CRYPTO_CMAC_CAST_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.5.1.5 CRYPTO_CMAC_CAST_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_CAST_Final_64(CRYPTO_CMAC_CAST_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 8 octets.

6.5.1.6 CRYPTO_CMAC_CAST_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_CAST_Init(      CRYPTO_CMAC_CAST_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.5.1.7 CRYPTO_CMAC_CAST_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_CAST_InitEx(      CRYPTO_CMAC_CAST_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.5.1.8 CRYPTO_CMAC_CAST_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_CAST_Kill(CRYPTO_CMAC_CAST_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.5.2 Generic API

The following table lists the CMAC-CAST functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_CAST_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_CAST_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_CAST_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_CAST_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_CAST_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_CAST_Kill()</code>	Destroy MAC context.

6.5.2.1 CRYPTO_MAC_CMAC_CAST_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.5.2.2 CRYPTO_MAC_CMAC_CAST_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_Final(void      * pContext,
                                 U8        * pMAC,
                                 unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.5.2.3 CRYPTO_MAC_CMAC_CAST_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_Final_64(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.5.2.4 CRYPTO_MAC_CMAC_CAST_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.5.2.5 CRYPTO_MAC_CMAC_CAST_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.5.2.6 CRYPTO_MAC_CMAC_CAST_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_CAST_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.6 CMAC-SEED

6.6.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

SEED is specified by the following document:

- IETF RFC 4269 — *The SEED Encryption Algorithm*

6.6.2 Type-safe API

The following table lists the CMAC-SEED type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_SEED_Calc()	Calculate MAC.
CRYPTO_CMAC_SEED_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_SEED_Init()	Initialize context.
CRYPTO_CMAC_SEED_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_SEED_Add()	Add data to MAC.
CRYPTO_CMAC_SEED_Final()	Finish MAC calculation.
CRYPTO_CMAC_SEED_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_SEED_Kill()	Destroy context.

6.6.2.1 CRYPTO_CMAC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_SEED_Add(          CRYPTO_CMAC_SEED_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.6.2.2 CRYPTO_CMAC_SEED_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_SEED_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned   KeyLen,
const U8      * pInput,
unsigned  InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.6.2.3 CRYPTO_CMAC_SEED_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_SEED_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.6.2.4 CRYPTO_CMAC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_SEED_Final(CRYPTO_CMAC_SEED_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.6.2.5 CRYPTO_CMAC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_SEED_Final_128(CRYPTO_CMAC_SEED_CONTEXT * pSelf,  
                                  U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.6.2.6 CRYPTO_CMAC_SEED_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_SEED_Init(      CRYPTO_CMAC_SEED_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.6.2.7 CRYPTO_CMAC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_SEED_InitEx(          CRYPTO_CMAC_SEED_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.6.2.8 CRYPTO_CMAC_SEED_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_SEED_Kill(CRYPTO_CMAC_SEED_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.6.3 Generic API

The following table lists the CMAC-SEED functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_SEED_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_SEED_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_SEED_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_SEED_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_SEED_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_SEED_Kill()</code>	Destroy MAC context.

6.6.3.1 CRYPTO_MAC_CMAC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.6.3.2 CRYPTO_MAC_CMAC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_Final(void      * pContext,
                                  U8        * pMAC,
                                  unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.6.3.3 CRYPTO_MAC_CMAC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.6.3.4 CRYPTO_MAC_CMAC_SEED_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.6.3.5 CRYPTO_MAC_CMAC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.6.3.6 CRYPTO_MAC_CMAC_SEED_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_SEED_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.7 CMAC-ARIA

6.7.1 Standards reference

CMAC is specified by the following document:

- *NIST SP 800-38B — Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

ARIA is specified by the following document:

- *IETF RFC 5794 — A Description of the ARIA Encryption Algorithm*

6.7.2 Type-safe API

The following table lists the CMAC-ARIA type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_ARIA_Calc()	Calculate MAC.
CRYPTO_CMAC_ARIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_ARIA_Init()	Initialize context.
CRYPTO_CMAC_ARIA_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_ARIA_Add()	Add data to MAC.
CRYPTO_CMAC_ARIA_Final()	Finish MAC calculation.
CRYPTO_CMAC_ARIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_ARIA_Kill()	Destroy context.

6.7.2.1 CRYPTO_CMAC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_ARIA_Add(          CRYPTO_CMAC_ARIA_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.7.2.2 CRYPTO_CMAC_ARIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_ARIA_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned   KeyLen,
const U8      * pInput,
unsigned  InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC, 16 octets.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to cipher key.
<code>KeyLen</code>	Octet length of the cipher key.
<code>pInput</code>	Pointer to message.
<code>InputLen</code>	Octet length of the message.

6.7.2.3 CRYPTO_CMAC_ARIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_ARIA_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.7.2.4 CRYPTO_CMAC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_ARIA_Final(CRYPTO_CMAC_ARIA_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.7.2.5 CRYPTO_CMAC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_ARIA_Final_128(CRYPTO_CMAC_ARIA_CONTEXT * pSelf,  
                                  U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.7.2.6 CRYPTO_CMAC_ARIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_ARIA_Init(      CRYPTO_CMAC_ARIA_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.7.2.7 CRYPTO_CMAC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_ARIA_InitEx(      CRYPTO_CMAC_ARIA_CONTEXT * pSelf,
                                     const U8                  * pKey,
                                     unsigned                 KeyLen,
                                     const U8                  * pIV,
                                     unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.7.2.8 CRYPTO_CMAC_ARIA_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_ARIA_Kill(CRYPTO_CMAC_ARIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.7.3 Generic API

The following table lists the CMAC-ARIA functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_ARIA_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_ARIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_ARIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_ARIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_ARIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_ARIA_Kill()</code>	Destroy MAC context.

6.7.3.1 CRYPTO_MAC_CMAC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.7.3.2 CRYPTO_MAC_CMAC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_Final(void      * pContext,
                                  U8        * pMAC,
                                  unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.7.3.3 CRYPTO_MAC_CMAC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.7.3.4 CRYPTO_MAC_CMAC_ARIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.7.3.5 CRYPTO_MAC_CMAC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.7.3.6 CRYPTO_MAC_CMAC_ARIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_ARIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.8 CMAC-Camellia

6.8.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

Camellia is specified by the following document:

- IETF RFC 3713 — *A Description of the Camellia Encryption Algorithm*

6.8.2 Type-safe API

The following table lists the CMAC-Camellia type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_CAMELLIA_Calc()	Calculate MAC.
CRYPTO_CMAC_CAMELLIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_CAMELLIA_Init()	Initialize context.
CRYPTO_CMAC_CAMELLIA_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_CAMELLIA_Add()	Add data to MAC.
CRYPTO_CMAC_CAMELLIA_Final()	Finish MAC calculation.
CRYPTO_CMAC_CAMELLIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_CAMELLIA_Kill()	Destroy context.

6.8.2.1 CRYPTO_CMAC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Add(          CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.8.2.2 CRYPTO_CMAC_CAMELLIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.8.2.3 CRYPTO_CMAC_CAMELLIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Calc_128(      U8      * pOutput,
                                           const U8      * pKey,
                                           unsigned      KeyLen,
                                           const U8      * pInput,
                                           unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.8.2.4 CRYPTO_CMAC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Final(CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf,  
                                  U8  
                                  unsigned  
                                  * pOutput,  
                                  OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.8.2.5 CRYPTO_CMAC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Final_128(CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf,  
                                      U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.8.2.6 CRYPTO_CMAC_CAMELLIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Init(          CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.8.2.7 CRYPTO_CMAC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_InitEx(      CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                  * pKey,
                                         unsigned                 KeyLen,
                                         const U8                  * pIV,
                                         unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.8.2.8 CRYPTO_CMAC_CAMELLIA_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_CAMELLIA_Kill(CRYPTO_CMAC_CAMELLIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.8.3 Generic API

The following table lists the CMAC-Camellia functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_CAMELLIA_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_CAMELLIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_CAMELLIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_CAMELLIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_CAMELLIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_CAMELLIA_Kill()</code>	Destroy MAC context.

6.8.3.1 CRYPTO_MAC_CMAC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.8.3.2 CRYPTO_MAC_CMAC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_Final(void      * pContext,
                                      U8        * pMAC,
                                      unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.8.3.3 CRYPTO_MAC_CMAC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.8.3.4 CRYPTO_MAC_CMAC_CAMELLIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.8.3.5 CRYPTO_MAC_CMAC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.8.3.6 CRYPTO_MAC_CMAC_CAMELLIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_CAMELLIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.9 CMAC-Twofish

6.9.1 Standards reference

CMAC is specified by the following document:

- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*

Twofish is specified by the following document:

- Schneier et al — *Twofish: A 128-Bit Block Cipher*

6.9.2 Type-safe API

The following table lists the CMAC-Twofish type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_CMAC_TWOFISH_Calc()</code>	Calculate MAC.
<code>CRYPTO_CMAC_TWOFISH_Calc_128()</code>	Calculate MAC, fixed size.
Incremental functions	
<code>CRYPTO_CMAC_TWOFISH_Init()</code>	Initialize context.
<code>CRYPTO_CMAC_TWOFISH_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_CMAC_TWOFISH_Add()</code>	Add data to MAC.
<code>CRYPTO_CMAC_TWOFISH_Final()</code>	Finish MAC calculation.
<code>CRYPTO_CMAC_TWOFISH_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_CMAC_TWOFISH_Kill()</code>	Destroy context.

6.9.2.1 CRYPTO_CMAC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Add(          CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf,
                                     const U8                  * pInput,
                                     unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.9.2.2 CRYPTO_CMAC_TWOFISH_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Calc(      U8      * pOutput,
                                      unsigned OutputLen,
                                      const U8   * pKey,
                                      unsigned KeyLen,
                                      const U8   * pInput,
                                      unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.9.2.3 CRYPTO_CMAC_TWOFISH_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.9.2.4 CRYPTO_CMAC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Final(CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf,  
                                U8  
                                unsigned  
                                * pOutput,  
                                OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.9.2.5 CRYPTO_CMAC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Final_128(CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf,  
                                     U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.9.2.6 CRYPTO_CMAC_TWOFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Init(      CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.9.2.7 CRYPTO_CMAC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_TWOFISH_InitEx( CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf ,  
                                  const U8 * pKey ,  
                                  unsigned KeyLen ,  
                                  const U8 * pIV ,  
                                  unsigned IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.9.2.8 CRYPTO_CMAC_TWOFISH_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_TWOFISH_Kill(CRYPTO_CMAC_TWOFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.9.3 Generic API

The following table lists the CMAC-Twofish functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_TWOFISH_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_TWOFISH_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_TWOFISH_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_TWOFISH_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_TWOFISH_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_TWOFISH_Kill()</code>	Destroy MAC context.

6.9.3.1 CRYPTO_MAC_CMAC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.9.3.2 CRYPTO_MAC_CMAC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_Final(void      * pContext,
                                     U8        * pMAC,
                                     unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.9.3.3 CRYPTO_MAC_CMAC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.9.3.4 CRYPTO_MAC_CMAC_TWOFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.9.3.5 CRYPTO_MAC_CMAC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.9.3.6 CRYPTO_MAC_CMAC_TWOFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_TWOFISH_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.10 CMAC-Blowfish

6.10.1 Type-safe API

The following table lists the CMAC-Blowfish type-safe API functions.

Function	Description
Message functions	
CRYPTO_CMAC_BLOWFISH_Calc()	Calculate MAC.
CRYPTO_CMAC_BLOWFISH_Calc_64()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_CMAC_BLOWFISH_Init()	Initialize context.
CRYPTO_CMAC_BLOWFISH_InitEx()	Initialize context, include subkey.
CRYPTO_CMAC_BLOWFISH_Add()	Add data to MAC.
CRYPTO_CMAC_BLOWFISH_Final()	Finish MAC calculation.
CRYPTO_CMAC_BLOWFISH_Final_64()	Finish MAC calculation, fixed size.
CRYPTO_CMAC_BLOWFISH_Kill()	Destroy context.

6.10.1.1 CRYPTO_CMAC_BLOWFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Add(          CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.10.1.2 CRYPTO_CMAC_BLOWFISH_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.10.1.3 CRYPTO_CMAC_BLOWFISH_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Calc_64(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.10.1.4 CRYPTO_CMAC_BLOWFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Final(CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf,  
                                U8                         * pOutput,  
                                unsigned                     OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.10.1.5 CRYPTO_CMAC_BLOWFISH_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Final_64(CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf,  
                                     U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 8 octets.

6.10.1.6 CRYPTO_CMAC_BLOWFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Init(          CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.10.1.7 CRYPTO_CMAC_BLOWFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_InitEx(      CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf,
                                         const U8                  * pKey,
                                         unsigned                 KeyLen,
                                         const U8                  * pIV,
                                         unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.10.1.8 CRYPTO_CMAC_BLOWFISH_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_BLOWFISH_Kill(CRYPTO_CMAC_BLOWFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.10.2 Generic API

The following table lists the CMAC-Blowfish functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_BLOWFISH_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_BLOWFISH_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_BLOWFISH_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_BLOWFISH_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_BLOWFISH_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_BLOWFISH_Kill()</code>	Destroy MAC context.

6.10.2.1 CRYPTO_MAC_CMAC_BLOWFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.10.2.2 CRYPTO_MAC_CMAC_BLOWFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_Final(void      * pContext,
                                      U8        * pMAC,
                                      unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.10.2.3 CRYPTO_MAC_CMAC_BLOWFISH_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_Final_64(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.10.2.4 CRYPTO_MAC_CMAC_BLOWFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.10.2.5 CRYPTO_MAC_CMAC_BLOWFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.10.2.6 CRYPTO_MAC_CMAC_BLOWFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_BLOWFISH_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.11 CMAC-PRESENT

6.11.1 Type-safe API

The following table lists the CMAC-PRESENT type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_CMAC_PRESENT_Calc()</code>	Calculate MAC.
<code>CRYPTO_CMAC_PRESENT_Calc_64()</code>	Calculate MAC, fixed size.
Incremental functions	
<code>CRYPTO_CMAC_PRESENT_Init()</code>	Initialize context.
<code>CRYPTO_CMAC_PRESENT_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_CMAC_PRESENT_Add()</code>	Add data to MAC.
<code>CRYPTO_CMAC_PRESENT_Final()</code>	Finish MAC calculation.
<code>CRYPTO_CMAC_PRESENT_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_CMAC_PRESENT_Kill()</code>	Destroy context.

6.11.1.1 CRYPTO_CMAC_PRESENT_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_CMAC_PRESENT_Add(          CRYPTO_CMAC_PRESENT_CONTEXT * pSelf,
                                     const U8                  * pInput,
                                     unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.11.1.2 CRYPTO_CMAC_PRESENT_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_CMAC_PRESENT_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.11.1.3 CRYPTO_CMAC_PRESENT_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_CMAC_PRESENT_Calc_64(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.11.1.4 CRYPTO_CMAC_PRESENT_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_CMAC_PRESENT_Final(CRYPTO_CMAC_PRESENT_CONTEXT * pSelf,  
                                U8  
                                unsigned  
                                * pOutput,  
                                OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.11.1.5 CRYPTO_CMAC_PRESENT_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_CMAC_PRESENT_Final_64(CRYPTO_CMAC_PRESENT_CONTEXT * pSelf,  
                                   U8                      * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 8 octets.

6.11.1.6 CRYPTO_CMAC_PRESENT_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_CMAC_PRESENT_Init(      CRYPTO_CMAC_PRESENT_CONTEXT * pSelf,
                                     const U8                  * pKey,
                                     unsigned                 KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.11.1.7 CRYPTO_CMAC_PRESENT_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_CMAC_PRESENT_InitEx( CRYPTO_CMAC_PRESENT_CONTEXT * pSelf ,  
                                  const U8 * pKey ,  
                                  unsigned KeyLen ,  
                                  const U8 * pIV ,  
                                  unsigned IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.11.1.8 CRYPTO_CMAC_PRESENT_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_CMAC_PRESENT_Kill(CRYPTO_CMAC_PRESENT_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.11.2 Generic API

The following table lists the CMAC-PRESENT functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_CMAC_PRESENT_Init()</code>	Initialize context.
<code>CRYPTO_MAC_CMAC_PRESENT_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_CMAC_PRESENT_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_CMAC_PRESENT_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_CMAC_PRESENT_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_CMAC_PRESENT_Kill()</code>	Destroy MAC context.

6.11.2.1 CRYPTO_MAC_CMAC_PRESENT_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.11.2.2 CRYPTO_MAC_CMAC_PRESENT_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_Final(void * pContext,  
                                     U8      * pMAC,  
                                     unsigned MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.11.2.3 CRYPTO_MAC_CMAC_PRESENT_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_Final_64(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.11.2.4 CRYPTO_MAC_CMAC_PRESENT_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.11.2.5 CRYPTO_MAC_CMAC_PRESENT_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.11.2.6 CRYPTO_MAC_CMAC_PRESENT_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_CMAC_PRESENT_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.12 GMAC-AES

6.12.1 Standards reference

GMAC is specified by the following document:

- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

AES is specified by the following document:

- FIPS PUB 197 — *Specification for the Advanced Encryption Standard (AES)*

6.12.2 Type-safe API

The following table lists the GMAC-AES type-safe API functions.

Function	Description
Message functions	
CRYPTO_GMAC_AES_Calc()	Calculate MAC.
CRYPTO_GMAC_AES_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_GMAC_AES_InitEx()	Initialize context, include subkey.
CRYPTO_GMAC_AES_Add()	Add data to MAC.
CRYPTO_GMAC_AES_Final()	Finish MAC calculation.
CRYPTO_GMAC_AES_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_GMAC_AES_Kill()	Destroy GMAC context.

6.12.2.1 CRYPTO_GMAC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_GMAC_AES_Add(      CRYPTO_GMAC_AES_CONTEXT * pSelf,
                                const U8             * pInput,
                                unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to input octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.12.2.2 CRYPTO_GMAC_AES_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_GMAC_AES_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pIV,
                                unsigned IVLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.12.2.3 CRYPTO_GMAC_AES_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_GMAC_AES_Calc_128(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pIV,
                                      unsigned      IVLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.12.2.4 CRYPTO_GMAC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_GMAC_AES_Final(CRYPTO_GMAC_AES_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.12.2.5 CRYPTO_GMAC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_GMAC_AES_Final_128(CRYPTO_GMAC_AES_CONTEXT * pSelf,  
                                U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 16 octets.

6.12.2.6 CRYPTO_GMAC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_GMAC_AES_InitEx(          CRYPTO_GMAC_AES_CONTEXT * pSelf ,  
                                const U8                  * pKey ,  
                                unsigned                 KeyLen ,  
                                const U8                  * pIV ,  
                                unsigned                 IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

6.12.2.7 CRYPTO_GMAC_AES_Kill()

Description

Destroy GMAC context.

Prototype

```
void CRYPTO_GMAC_AES_Kill(CRYPTO_GMAC_AES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.12.3 Generic API

The following table lists the GMAC-AES functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_GMAC_AES_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_GMAC_AES_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_GMAC_AES_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_GMAC_AES_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_GMAC_AES_Kill()</code>	Destroy MAC context.

6.12.3.1 CRYPTO_MAC_GMAC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_GMAC_AES_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.12.3.2 CRYPTO_MAC_GMAC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_GMAC_AES_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.12.3.3 CRYPTO_MAC_GMAC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_GMAC_AES_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.12.3.4 CRYPTO_MAC_GMAC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_GMAC_AES_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.12.3.5 CRYPTO_MAC_GMAC_AES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_GMAC_AES_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.13 GMAC-SEED

6.13.1 Standards reference

GMAC is specified by the following document:

- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

SEED is specified by the following document:

- IETF RFC 4269 — *The SEED Encryption Algorithm*

6.13.2 Type-safe API

The following table lists the GMAC-SEED type-safe API functions.

Function	Description
Message functions	
CRYPTO_GMAC_SEED_Calc()	Calculate MAC.
CRYPTO_GMAC_SEED_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_GMAC_SEED_InitEx()	Initialize context, include subkey.
CRYPTO_GMAC_SEED_Add()	Add data to MAC.
CRYPTO_GMAC_SEED_Final()	Finish MAC calculation.
CRYPTO_GMAC_SEED_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_GMAC_SEED_Kill()	Destroy GMAC context.

6.13.2.1 CRYPTO_GMAC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_GMAC_SEED_Add(          CRYPTO_GMAC_SEED_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to input octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.13.2.2 CRYPTO_GMAC_SEED_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_GMAC_SEED_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned KeyLen,
const U8      * pIV,
unsigned IVLen,
const U8      * pInput,
unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.13.2.3 CRYPTO_GMAC_SEED_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_GMAC_SEED_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pIV,
                                         unsigned      IVLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.13.2.4 CRYPTO_GMAC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_GMAC_SEED_Final(CRYPTO_GMAC_SEED_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.13.2.5 CRYPTO_GMAC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_GMAC_SEED_Final_128(CRYPTO_GMAC_SEED_CONTEXT * pSelf,  
                                  U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 16 octets.

6.13.2.6 CRYPTO_GMAC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_GMAC_SEED_InitEx(      CRYPTO_GMAC_SEED_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

6.13.2.7 CRYPTO_GMAC_SEED_Kill()

Description

Destroy GMAC context.

Prototype

```
void CRYPTO_GMAC_SEED_Kill(CRYPTO_GMAC_SEED_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.13.3 Generic API

The following table lists the GMAC-SEED functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_GMAC_SEED_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_GMAC_SEED_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_GMAC_SEED_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_GMAC_SEED_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_GMAC_SEED_Kill()</code>	Destroy MAC context.

6.13.3.1 CRYPTO_MAC_GMAC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_GMAC_SEED_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.13.3.2 CRYPTO_MAC_GMAC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_GMAC_SEED_Final(void      * pContext,
                                 U8        * pMAC,
                                 unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.13.3.3 CRYPTO_MAC_GMAC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_GMAC_SEED_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.13.3.4 CRYPTO_MAC_GMAC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_GMAC_SEED_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.13.3.5 CRYPTO_MAC_GMAC_SEED_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_GMAC_SEED_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.14 GMAC-ARIA

6.14.1 Standards reference

GMAC is specified by the following document:

- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

ARIA is specified by the following document:

- IETF RFC 5794 — *A Description of the ARIA Encryption Algorithm*

6.14.2 Type-safe API

The following table lists the GMAC-ARIA type-safe API functions.

Function	Description
Message functions	
CRYPTO_GMAC_ARIA_Calc()	Calculate MAC.
CRYPTO_GMAC_ARIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_GMAC_ARIA_InitEx()	Initialize context, include subkey.
CRYPTO_GMAC_ARIA_Add()	Add data to MAC.
CRYPTO_GMAC_ARIA_Final()	Finish MAC calculation.
CRYPTO_GMAC_ARIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_GMAC_ARIA_Kill()	Destroy GMAC context.

6.14.2.1 CRYPTO_GMAC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_GMAC_ARIA_Add(          CRYPTO_GMAC_ARIA_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to input octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.14.2.2 CRYPTO_GMAC_ARIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_GMAC_ARIA_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned KeyLen,
const U8      * pIV,
unsigned IVLen,
const U8      * pInput,
unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.14.2.3 CRYPTO_GMAC_ARIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_GMAC_ARIA_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pIV,
                                         unsigned      IVLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.14.2.4 CRYPTO_GMAC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_GMAC_ARIA_Final(CRYPTO_GMAC_ARIA_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.14.2.5 CRYPTO_GMAC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_GMAC_ARIA_Final_128(CRYPTO_GMAC_ARIA_CONTEXT * pSelf,  
                                  U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 16 octets.

6.14.2.6 CRYPTO_GMAC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_GMAC_ARIA_InitEx(      CRYPTO_GMAC_ARIA_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

6.14.2.7 CRYPTO_GMAC_ARIA_Kill()

Description

Destroy GMAC context.

Prototype

```
void CRYPTO_GMAC_ARIA_Kill(CRYPTO_GMAC_ARIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.14.3 Generic API

The following table lists the GMAC-ARIA functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_GMAC_ARIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_GMAC_ARIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_GMAC_ARIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_GMAC_ARIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_GMAC_ARIA_Kill()</code>	Destroy MAC context.

6.14.3.1 CRYPTO_MAC_GMAC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_GMAC_ARIA_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.14.3.2 CRYPTO_MAC_GMAC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_GMAC_ARIA_Final(void      * pContext,
                                  U8        * pMAC,
                                  unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.14.3.3 CRYPTO_MAC_GMAC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_GMAC_ARIA_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.14.3.4 CRYPTO_MAC_GMAC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_GMAC_ARIA_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.14.3.5 CRYPTO_MAC_GMAC_ARIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_GMAC_ARIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.15 GMAC-Camellia

6.15.1 Standards reference

GMAC is specified by the following document:

- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

Camellia is specified by the following document:

- IETF RFC 3713 — *A Description of the Camellia Encryption Algorithm*

6.15.2 Type-safe API

The following table lists the GMAC-Camellia type-safe API functions.

Function	Description
Message functions	
CRYPTO_GMAC_CAMELLIA_Calc()	Calculate MAC.
CRYPTO_GMAC_CAMELLIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_GMAC_CAMELLIA_InitEx()	Initialize context, include subkey.
CRYPTO_GMAC_CAMELLIA_Add()	Add data to MAC.
CRYPTO_GMAC_CAMELLIA_Final()	Finish MAC calculation.
CRYPTO_GMAC_CAMELLIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_GMAC_CAMELLIA_Kill()	Destroy GMAC context.

6.15.2.1 CRYPTO_GMAC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Add(          CRYPTO_GMAC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to input octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.15.2.2 CRYPTO_GMAC_CAMELLIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Calc(      U8      * pOutput,
                                       unsigned   OutputLen,
                                       const U8    * pKey,
                                       unsigned   KeyLen,
                                       const U8    * pIV,
                                       unsigned   IVLen,
                                       const U8    * pInput,
                                       unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.15.2.3 CRYPTO_GMAC_CAMELLIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Calc_128(      U8      * pOutput,
                                           const U8      * pKey,
                                           unsigned      KeyLen,
                                           const U8      * pIV,
                                           unsigned      IVLen,
                                           const U8      * pInput,
                                           unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.15.2.4 CRYPTO_GMAC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Final(CRYPTO_GMAC_CAMELLIA_CONTEXT * pSelf,  
                                  U8                         * pOutput,  
                                  unsigned                     OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.15.2.5 CRYPTO_GMAC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Final_128(CRYPTO_GMAC_CAMELLIA_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 16 octets.

6.15.2.6 CRYPTO_GMAC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_InitEx(      CRYPTO_GMAC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                  * pKey,
                                         unsigned                 KeyLen,
                                         const U8                  * pIV,
                                         unsigned                 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

6.15.2.7 CRYPTO_GMAC_CAMELLIA_Kill()

Description

Destroy GMAC context.

Prototype

```
void CRYPTO_GMAC_CAMELLIA_Kill(CRYPTO_GMAC_CAMELLIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.15.3 Generic API

The following table lists the GMAC-Camellia functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_GMAC_CAMELLIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_GMAC_CAMELLIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_GMAC_CAMELLIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_GMAC_CAMELLIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_GMAC_CAMELLIA_Kill()</code>	Destroy MAC context.

6.15.3.1 CRYPTO_MAC_GMAC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_GMAC_CAMELLIA_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.15.3.2 CRYPTO_MAC_GMAC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_GMAC_CAMELLIA_Final(void      * pContext,
                                      U8        * pMAC,
                                      unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.15.3.3 CRYPTO_MAC_GMAC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_GMAC_CAMELLIA_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.15.3.4 CRYPTO_MAC_GMAC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_GMAC_CAMELLIA_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.15.3.5 CRYPTO_MAC_GMAC_CAMELLIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_GMAC_CAMELLIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.16 GMAC-Twofish

6.16.1 Standards reference

GMAC is specified by the following document:

- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

Twofish is specified by the following document:

- Schneier et al — *Twofish: A 128-Bit Block Cipher*

6.16.2 Type-safe API

The following table lists the GMAC-Twofish type-safe API functions.

Function	Description
Message functions	
CRYPTO_GMAC_TWOFISH_Calc()	Calculate MAC.
CRYPTO_GMAC_TWOFISH_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_GMAC_TWOFISH_InitEx()	Initialize context, include subkey.
CRYPTO_GMAC_TWOFISH_Add()	Add data to MAC.
CRYPTO_GMAC_TWOFISH_Final()	Finish MAC calculation.
CRYPTO_GMAC_TWOFISH_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_GMAC_TWOFISH_Kill()	Destroy GMAC context.

6.16.2.1 CRYPTO_GMAC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Add(          CRYPTO_GMAC_TWOFISH_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to input octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.16.2.2 CRYPTO_GMAC_TWOFISH_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pIV,
                                     unsigned IVLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.16.2.3 CRYPTO_GMAC_TWOFISH_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pIV,
                                         unsigned     IVLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.16.2.4 CRYPTO_GMAC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Final(CRYPTO_GMAC_TWOFISH_CONTEXT * pSelf,  
                                U8  
                                unsigned  
                                * pOutput,  
                                OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.16.2.5 CRYPTO_GMAC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Final_128(CRYPTO_GMAC_TWOFISH_CONTEXT * pSelf,  
                                     U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 16 octets.

6.16.2.6 CRYPTO_GMAC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_GMAC_TWOFISH_InitEx(          CRYPTO_GMAC_TWOFISH_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

6.16.2.7 CRYPTO_GMAC_TWOFISH_Kill()

Description

Destroy GMAC context.

Prototype

```
void CRYPTO_GMAC_TWOFISH_Kill(CRYPTO_GMAC_TWOFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.16.3 Generic API

The following table lists the GMAC-Twofish functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_GMAC_TWOFISH_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_GMAC_TWOFISH_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_GMAC_TWOFISH_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_GMAC_TWOFISH_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_GMAC_TWOFISH_Kill()</code>	Destroy MAC context.

6.16.3.1 CRYPTO_MAC_GMAC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_GMAC_TWOFISH_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.16.3.2 CRYPTO_MAC_GMAC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_GMAC_TWOFISH_Final(void      * pContext,
                                     U8        * pMAC,
                                     unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.16.3.3 CRYPTO_MAC_GMAC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_GMAC_TWOFISH_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.16.3.4 CRYPTO_MAC_GMAC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_GMAC_TWOFISH_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.16.3.5 CRYPTO_MAC_GMAC_TWOFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_GMAC_TWOFISH_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.17 HMAC-MD5

6.17.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

MD5 is specified by the following document:

- IETF RFC 1321 — *The MD5 Message-Digest Algorithm*

6.17.2 Type-safe API

The following table lists the HMAC-MD5 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_HMAC_MD5_Calc()</code>	Calculate MAC.
<code>CRYPTO_HMAC_MD5_Calc_160()</code>	Calculate MAC, fixed size.
Incremental functions	
<code>CRYPTO_HMAC_MD5_Init()</code>	Initialize context.
<code>CRYPTO_HMAC_MD5_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_HMAC_MD5_Add()</code>	Add data to MAC.
<code>CRYPTO_HMAC_MD5_Final()</code>	Finalize MAC calculation.
<code>CRYPTO_HMAC_MD5_Final_160()</code>	Finalize MAC calculation, fixed size.
<code>CRYPTO_HMAC_MD5_Kill()</code>	Destroy HMAC context.
<code>CRYPTO_HMAC_MD5_Reset()</code>	Reset MAC to initial state.

6.17.2.1 CRYPTO_HMAC_MD5_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_MD5_Add(      CRYPTO_HMAC_MD5_CONTEXT * pSelf,
                                const U8             * pInput,
                                unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-MD5 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.17.2.2 CRYPTO_HMAC_MD5_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_MD5_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.17.2.3 CRYPTO_HMAC_MD5_Calc_160()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_MD5_Calc_160(      U8      * pOutput,
                                       const U8      * pKey,
                                       unsigned      KeyLen,
                                       const U8      * pInput,
                                       unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 20 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.17.2.4 CRYPTO_HMAC_MD5_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_MD5_Final(CRYPTO_HMAC_MD5_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-MD5 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.17.2.5 CRYPTO_HMAC_MD5_Final_160()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_MD5_Final_160(CRYPTO_HMAC_MD5_CONTEXT * pSelf,  
                                U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-MD5 context.
pOutput	Pointer to object that receives the MAC, 20 octets.

6.17.2.6 CRYPTO_HMAC_MD5_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_MD5_Init(      CRYPTO_HMAC_MD5_CONTEXT * pSelf,
                                const U8                 * pKey,
                                unsigned                KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-MD5 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.17.2.7 CRYPTO_HMAC_MD5_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_MD5_InitEx(          CRYPTO_HMAC_MD5_CONTEXT * pSelf ,  
                                const U8                      * pKey ,  
                                unsigned                     KeyLen ,  
                                const U8                      * pIV ,  
                                unsigned                     IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.17.2.8 CRYPTO_HMAC_MD5_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_MD5_Kill(CRYPTO_HMAC_MD5_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.17.2.9 CRYPTO_HMAC_MD5_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_MD5_Reset(CRYPTO_HMAC_MD5_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-MD5 context.

6.17.3 Generic API

The following table lists the HMAC-MD5 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_MD5_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_MD5_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_MD5_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_MD5_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_MD5_Final_96()</code>	Finish computation of the HMAC-MD5-96 HMAC and write to the output buffer.
<code>CRYPTO_MAC_HMAC_MD5_Final_160()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_MD5_Kill()</code>	Destroy MAC context.

6.17.3.1 CRYPTO_MAC_HMAC_MD5_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.17.3.2 CRYPTO_MAC_HMAC_MD5_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Final(void * pContext,  
                               U8      * pMAC,  
                               unsigned MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.17.3.3 CRYPTO_MAC_HMAC_MD5_Final_96()

Description

Finish computation of the HMAC-MD5-96 HMAC and write to the output buffer.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Final_96(void * pSelf,  
                                  U8     * pMAC);
```

Parameters

Parameter	Description
pSelf	HMAC-MD5 context.
pMAC	Pointer to object that receives MAC of CRYPTO_MD5_96_DIGEST_BYTE_COUNT octets.

6.17.3.4 CRYPTO_MAC_HMAC_MD5_Final_160()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Final_160(void * pContext,  
                                     U8     * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.17.3.5 CRYPTO_MAC_HMAC_MD5_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Init(      void      * pContext,
                                     const U8      * pKey,
                                     unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.17.3.6 CRYPTO_MAC_HMAC_MD5_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.17.3.7 CRYPTO_MAC_HMAC_MD5_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_MD5_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.18 HMAC-RIPEMD-160

6.18.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

MD5 is specified by the following document:

- Antoon Bosselaers — *The hash function RIPEMD-160*

6.18.2 Type-safe API

The following table lists the HMAC-RIPEMD-160 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_RIPEMD160_Calc()	Calculate MAC.
CRYPTO_HMAC_RIPEMD160_Calc_160()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_RIPEMD160_Init()	Initialize context.
CRYPTO_HMAC_RIPEMD160_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_RIPEMD160_Add()	Add data to MAC.
CRYPTO_HMAC_RIPEMD160_Final()	Finalize MAC calculation.
CRYPTO_HMAC_RIPEMD160_Final_160()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_RIPEMD160_Kill()	Destroy HMAC context.
CRYPTO_HMAC_RIPEMD160_Reset()	Reset MAC to initial state.

6.18.2.1 CRYPTO_HMAC_RIPEMD160_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Add(          CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-RIPEMD-160 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.18.2.2 CRYPTO_HMAC_RIPEMD160_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.18.2.3 CRYPTO_HMAC_RIPEMD160_Calc_160()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Calc_160(      U8      * pOutput,
                                             const U8      * pKey,
                                             unsigned      KeyLen,
                                             const U8      * pInput,
                                             unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 20 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.18.2.4 CRYPTO_HMAC_RIPEMD160_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Final(CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf,  
                                    U8 * pOutput,  
                                    unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-RIPEMD-160 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.18.2.5 CRYPTO_HMAC_RIPEMD160_Final_160()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Final_160(CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-RIPEMD-160 context.
pOutput	Pointer to object that receives the MAC, 20 octets.

6.18.2.6 CRYPTO_HMAC_RIPEMD160_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Init(          CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-RIPEMD-160 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.18.2.7 CRYPTO_HMAC_RIPEMD160_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_InitEx(      CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf,
                                           const U8                      * pKey,
                                           unsigned                     KeyLen,
                                           const U8                      * pIV,
                                           unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.18.2.8 CRYPTO_HMAC_RIPEMD160_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Kill(CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MAC context.

6.18.2.9 CRYPTO_HMAC_RIPEMD160_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_RIPEMD160_Reset(CRYPTO_HMAC_RIPEMD160_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to HMAC-RIPEMD160 context.

6.18.3 Generic API

The following table lists the HMAC-RIPEMD-160 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_RIPEMD160_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_RIPEMD160_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_RIPEMD160_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_RIPEMD160_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_RIPEMD160_Final_160()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_RIPEMD160_Kill()</code>	Destroy MAC context.

6.18.3.1 CRYPTO_MAC_HMAC_RIPEMD160_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_Add(      void      * pContext,
                                             const U8      * pInput,
                                             unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.18.3.2 CRYPTO_MAC_HMAC_RIPEMD160_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_Final(void * pContext,  
                                      U8      * pMAC,  
                                      unsigned MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.18.3.3 CRYPTO_MAC_HMAC_RIPEMD160_Final_160()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_Final_160(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.18.3.4 CRYPTO_MAC_HMAC_RIPEMD160_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_Init(      void      * pContext,
                                             const U8      * pKey,
                                             unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.18.3.5 CRYPTO_MAC_HMAC_RIPEMD160_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.18.3.6 CRYPTO_MAC_HMAC_RIPEMD160_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_RIPEMD160_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.19 HMAC-SHA-1

6.19.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-1 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.19.2 Type-safe API

The following table lists the HMAC-SHA-1 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA1_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA1_Calc_160()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA1_Init()	Initialize context.
CRYPTO_HMAC_SHA1_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA1_Add()	Add data to MAC.
CRYPTO_HMAC_SHA1_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA1_Final_160()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA1_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA1_Reset()	Reset MAC to initial state.

6.19.2.1 CRYPTO_HMAC_SHA1_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA1_Add(      CRYPTO_HMAC_SHA1_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-1 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.19.2.2 CRYPTO_HMAC_SHA1_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA1_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
                                  unsigned KeyLen,
const U8      * pInput,
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.19.2.3 CRYPTO_HMAC_SHA1_Calc_160()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA1_Calc_160(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 20 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.19.2.4 CRYPTO_HMAC_SHA1_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA1_Final(CRYPTO_HMAC_SHA1_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-1 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.19.2.5 CRYPTO_HMAC_SHA1_Final_160()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA1_Final_160(CRYPTO_HMAC_SHA1_CONTEXT * pSelf,  
                                  U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-1 context.
pOutput	Pointer to object that receives the MAC, 20 octets.

6.19.2.6 CRYPTO_HMAC_SHA1_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA1_Init(          CRYPTO_HMAC_SHA1_CONTEXT * pSelf,
                                  const U8                      * pKey,
                                  unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-1 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.19.2.7 CRYPTO_HMAC_SHA1_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA1_InitEx(      CRYPTO_HMAC_SHA1_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.19.2.8 CRYPTO_HMAC_SHA1_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA1_Kill(CRYPTO_HMAC_SHA1_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.19.2.9 CRYPTO_HMAC_SHA1_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA1_Reset(CRYPTO_HMAC_SHA1_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA1 context.

6.19.3 Generic API

The following table lists the HMAC-SHA-1 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA1_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA1_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA1_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA1_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA1_Final_96()</code>	Finish computation of the HMAC-SHA1-96 HMAC and write to the output buffer.
<code>CRYPTO_MAC_HMAC_SHA1_Final_160()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA1_Kill()</code>	Destroy MAC context.

6.19.3.1 CRYPTO_MAC_HMAC_SHA1_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.19.3.2 CRYPTO_MAC_HMAC_SHA1_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.19.3.3 CRYPTO_MAC_HMAC_SHA1_Final_96()

Description

Finish computation of the HMAC-SHA1-96 HMAC and write to the output buffer.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Final_96(void * pSelf,  
                                    U8      * pMAC);
```

Parameters

Parameter	Description
pSelf	HMAC-SHA1 context.
pMAC	Pointer to object that receives MAC of CRYPTO_SHA1_96_DIGEST_BYTE_COUNT octets.

6.19.3.4 CRYPTO_MAC_HMAC_SHA1_Final_160()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Final_160(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.19.3.5 CRYPTO_MAC_HMAC_SHA1_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Init(      void      * pContext,
                                      const U8      * pKey,
                                      unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.19.3.6 CRYPTO_MAC_HMAC_SHA1_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.19.3.7 CRYPTO_MAC_HMAC_SHA1_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA1_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.19.4 Self-test API

The following table lists the HMAC-SHA-1 self-test API functions.

Function	Description
<code>CRYPTO_HMAC_SHA1_RFC2202_SelfTest()</code>	Run all RFC 2202 HMAC-SHA-1 test vectors.
<code>CRYPTO_HMAC_SHA1_CAVS_SelfTest()</code>	Run AES-CMAC self-test.

6.19.4.1 CRYPTO_HMAC_SHA1_RFC2202_SelfTest()

Description

Run all RFC 2202 HMAC-SHA-1 test vectors.

Prototype

```
void CRYPTO_HMAC_SHA1_RFC2202_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.19.4.2 CRYPTO_HMAC_SHA1_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_HMAC_SHA1_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.20 HMAC-SHA-224

6.20.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-224 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.20.2 Type-safe API

The following table lists the HMAC-SHA-224 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA224_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA224_Calc_224()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA224_Init()	Initialize context.
CRYPTO_HMAC_SHA224_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA224_Add()	Add data to MAC.
CRYPTO_HMAC_SHA224_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA224_Final_224()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA224_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA224_Reset()	Reset MAC to initial state.

6.20.2.1 CRYPTO_HMAC_SHA224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA224_Add(          CRYPTO_HMAC_SHA224_CONTEXT * pSelf,
                                  const U8                      * pInput,
                                  unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-224 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.20.2.2 CRYPTO_HMAC_SHA224_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA224_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.20.2.3 CRYPTO_HMAC_SHA224_Calc_224()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA224_Calc_224(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 28 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.20.2.4 CRYPTO_HMAC_SHA224_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA224_Final(CRYPTO_HMAC_SHA224_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-224 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.20.2.5 CRYPTO_HMAC_SHA224_Final_224()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA224_Final_224(CRYPTO_HMAC_SHA224_CONTEXT * pSelf,  
                                     U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-224 context.
pOutput	Pointer to object that receives the MAC, 28 octets.

6.20.2.6 CRYPTO_HMAC_SHA224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA224_Init(          CRYPTO_HMAC_SHA224_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-224 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.20.2.7 CRYPTO_HMAC_SHA224_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA224_InitEx(      CRYPTO_HMAC_SHA224_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.20.2.8 CRYPTO_HMAC_SHA224_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA224_Kill(CRYPTO_HMAC_SHA224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MAC context.

6.20.2.9 CRYPTO_HMAC_SHA224_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA224_Reset(CRYPTO_HMAC_SHA224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA224 context.

6.20.3 Generic API

The following table lists the HMAC-SHA-224 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA224_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA224_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA224_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA224_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA224_Final_224()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA224_Kill()</code>	Destroy MAC context.

6.20.3.1 CRYPTO_MAC_HMAC_SHA224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
<code>pContext</code>	Pointer to MAC context.
<code>pInput</code>	Pointer to input to add to MAC.
<code>InputLen</code>	Octet length of the input string.

6.20.3.2 CRYPTO_MAC_HMAC_SHA224_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_Final(void      * pContext,
                                    U8        * pMAC,
                                    unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.20.3.3 CRYPTO_MAC_HMAC_SHA224_Final_224()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_Final_224(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.20.3.4 CRYPTO_MAC_HMAC_SHA224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.20.3.5 CRYPTO_MAC_HMAC_SHA224_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.20.3.6 CRYPTO_MAC_HMAC_SHA224_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.20.4 Self-test API

The following table lists the HMAC-SHA-224 self-test API functions.

Function	Description
CRYPTO_HMAC_SHA224_CAVS_SelfTest()	Run AES-CMAC self-test.

6.20.4.1 CRYPTO_HMAC_SHA224_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_HMAC_SHA224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.21 HMAC-SHA-256

6.21.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-256 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.21.2 Type-safe API

The following table lists the HMAC-SHA-256 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA256_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA256_Calc_256()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA256_Init()	Initialize context.
CRYPTO_HMAC_SHA256_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA256_Add()	Add data to MAC.
CRYPTO_HMAC_SHA256_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA256_Final_256()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA256_Reset()	Reset MAC to initial state.
CRYPTO_HMAC_SHA256_Kill()	Destroy HMAC context.

6.21.2.1 CRYPTO_HMAC_SHA256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA256_Add(          CRYPTO_HMAC_SHA256_CONTEXT * pSelf,
                                  const U8                      * pInput,
                                  unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-256 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.21.2.2 CRYPTO_HMAC_SHA256_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA256_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.21.2.3 CRYPTO_HMAC_SHA256_Calc_256()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA256_Calc_256(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 32 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.21.2.4 CRYPTO_HMAC_SHA256_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA256_Final(CRYPTO_HMAC_SHA256_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-256 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.21.2.5 CRYPTO_HMAC_SHA256_Final_256()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA256_Final_256(CRYPTO_HMAC_SHA256_CONTEXT * pSelf,  
                                    U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-256 context.
pOutput	Pointer to object that receives the MAC, 32 octets.

6.21.2.6 CRYPTO_HMAC_SHA256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA256_Init(          CRYPTO_HMAC_SHA256_CONTEXT * pSelf,
                                     const U8                  * pKey,
                                     unsigned                 KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-256 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.21.2.7 CRYPTO_HMAC_SHA256_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA256_InitEx(      CRYPTO_HMAC_SHA256_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.21.2.8 CRYPTO_HMAC_SHA256_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA256_Reset(CRYPTO_HMAC_SHA256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA256 context.

6.21.2.9 CRYPTO_HMAC_SHA256_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA256_Kill(CRYPTO_HMAC_SHA256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.21.3 Generic API

The following table lists the HMAC-SHA-256 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA256_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA256_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA256_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA256_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA256_Final_256()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA256_Kill()</code>	Destroy MAC context.

6.21.3.1 CRYPTO_MAC_HMAC_SHA256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
<code>pContext</code>	Pointer to MAC context.
<code>pInput</code>	Pointer to input to add to MAC.
<code>InputLen</code>	Octet length of the input string.

6.21.3.2 CRYPTO_MAC_HMAC_SHA256_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_Final(void      * pContext,
                                    U8        * pMAC,
                                    unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.21.3.3 CRYPTO_MAC_HMAC_SHA256_Final_256()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_Final_256(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.21.3.4 CRYPTO_MAC_HMAC_SHA256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.21.3.5 CRYPTO_MAC_HMAC_SHA256_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.21.3.6 CRYPTO_MAC_HMAC_SHA256_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.21.4 Self-test API

The following table lists the HMAC-SHA-256 self-test API functions.

Function	Description
CRYPTO_HMAC_SHA256_CAVS_SelfTest()	Run AES-CMAC self-test.
CRYPTO_HMAC_SHA256_RFC4231_SelfTest()	Run all RFC 4231 HMAC-SHA-256 test vectors.

6.21.4.1 CRYPTO_HMAC_SHA256_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_HMAC_SHA256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.21.4.2 CRYPTO_HMAC_SHA256_RFC4231_SelfTest()

Description

Run all RFC 4231 HMAC-SHA-256 test vectors.

Prototype

```
void CRYPTO_HMAC_SHA256_RFC4231_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.22 HMAC-SHA-384

6.22.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-384 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.22.2 Type-safe API

The following table lists the HMAC-SHA-384 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA384_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA384_Calc_384()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA384_Add()	Add data to MAC.
CRYPTO_HMAC_SHA384_Init()	Initialize context.
CRYPTO_HMAC_SHA384_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA384_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA384_Final_384()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA384_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA384_Reset()	Reset MAC to initial state.

6.22.2.1 CRYPTO_HMAC_SHA384_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA384_Add(          CRYPTO_HMAC_SHA384_CONTEXT * pSelf,
                                  const U8                      * pInput,
                                  unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-384 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.22.2.2 CRYPTO_HMAC_SHA384_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA384_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.22.2.3 CRYPTO_HMAC_SHA384_Calc_384()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA384_Calc_384(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 48 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.22.2.4 CRYPTO_HMAC_SHA384_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA384_Final(CRYPTO_HMAC_SHA384_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-384 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.22.2.5 CRYPTO_HMAC_SHA384_Final_384()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA384_Final_384(CRYPTO_HMAC_SHA384_CONTEXT * pSelf,  
                                    U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-384 context.
pOutput	Pointer to object that receives the MAC, 48 octets.

6.22.2.6 CRYPTO_HMAC_SHA384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA384_Init(          CRYPTO_HMAC_SHA384_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-384 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.22.2.7 CRYPTO_HMAC_SHA384_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA384_InitEx(      CRYPTO_HMAC_SHA384_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.22.2.8 CRYPTO_HMAC_SHA384_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA384_Kill(CRYPTO_HMAC_SHA384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MAC context.

6.22.2.9 CRYPTO_HMAC_SHA384_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA384_Reset(CRYPTO_HMAC_SHA384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA384 context.

6.22.3 Generic API

The following table lists the HMAC-SHA-384 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA384_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA384_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA384_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA384_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA384_Final_384()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA384_Kill()</code>	Destroy MAC context.

6.22.3.1 CRYPTO_MAC_HMAC_SHA384_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_Add(      void      * pContext,
                                         const U8     * pInput,
                                         unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.22.3.2 CRYPTO_MAC_HMAC_SHA384_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_Final(void      * pContext,
                                    U8        * pMAC,
                                    unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.22.3.3 CRYPTO_MAC_HMAC_SHA384_Final_384()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_Final_384(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.22.3.4 CRYPTO_MAC_HMAC_SHA384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.22.3.5 CRYPTO_MAC_HMAC_SHA384_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.22.3.6 CRYPTO_MAC_HMAC_SHA384_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA384_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.22.4 Self-test API

The following table lists the HMAC-SHA-384 self-test API functions.

Function	Description
CRYPTO_HMAC_SHA384_CAVS_SelfTest()	Run AES-CMAC self-test.

6.22.4.1 CRYPTO_HMAC_SHA384_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_HMAC_SHA384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.23 HMAC-SHA-512

6.23.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-512 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.23.2 Type-safe API

The following table lists the HMAC-SHA-512 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA512_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA512_Calc_512()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA512_Add()	Add data to MAC.
CRYPTO_HMAC_SHA512_Init()	Initialize context.
CRYPTO_HMAC_SHA512_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA512_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA512_Final_512()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA512_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA512_Reset()	Reset MAC to initial state.

6.23.2.1 CRYPTO_HMAC_SHA512_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_Add(          CRYPTO_HMAC_SHA512_CONTEXT * pSelf,
                                  const U8                      * pInput,
                                  unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.23.2.2 CRYPTO_HMAC_SHA512_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.23.2.3 CRYPTO_HMAC_SHA512_Calc_512()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_Calc_512(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 64 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.23.2.4 CRYPTO_HMAC_SHA512_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA512_Final(CRYPTO_HMAC_SHA512_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.23.2.5 CRYPTO_HMAC_SHA512_Final_512()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_Final_512(CRYPTO_HMAC_SHA512_CONTEXT * pSelf,  
                                     U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512 context.
pOutput	Pointer to object that receives the MAC, 64 octets.

6.23.2.6 CRYPTO_HMAC_SHA512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA512_Init(          CRYPTO_HMAC_SHA512_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.23.2.7 CRYPTO_HMAC_SHA512_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA512_InitEx(      CRYPTO_HMAC_SHA512_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.23.2.8 CRYPTO_HMAC_SHA512_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA512_Kill(CRYPTO_HMAC_SHA512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MAC context.

6.23.2.9 CRYPTO_HMAC_SHA512_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA512_Reset(CRYPTO_HMAC_SHA512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA512 context.

6.23.3 Generic API

The following table lists the HMAC-SHA-512 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA512_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA512_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA512_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA512_Final_512()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA512_Kill()</code>	Destroy MAC context.

6.23.3.1 CRYPTO_MAC_HMAC_SHA512_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.23.3.2 CRYPTO_MAC_HMAC_SHA512_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_Final(void      * pContext,
                                    U8        * pMAC,
                                    unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.23.3.3 CRYPTO_MAC_HMAC_SHA512_Final_512()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_Final_512(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.23.3.4 CRYPTO_MAC_HMAC_SHA512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.23.3.5 CRYPTO_MAC_HMAC_SHA512_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.23.4 Self-test API

The following table lists the HMAC-SHA-512 self-test API functions.

Function	Description
CRYPTO_HMAC_SHA512_CAVS_SelfTest()	Run AES-CMAC self-test.
CRYPTO_HMAC_SHA512_RFC4231_SelfTest()	Run all HMAC-SHA-512 RFC 4231 test vectors.

6.23.4.1 CRYPTO_HMAC_SHA512_CAVS_SelfTest()

Description

Run AES-CMAC self-test.

Prototype

```
void CRYPTO_HMAC_SHA512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.23.4.2 CRYPTO_HMAC_SHA512_RFC4231_SelfTest()

Description

Run all HMAC-SHA-512 RFC 4231 test vectors.

Prototype

```
void CRYPTO_HMAC_SHA512_RFC4231_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.24 HMAC-SHA-512/224

6.24.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-512/224 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.24.2 Type-safe API

The following table lists the HMAC-SHA-512/224 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA512_224_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA512_224_Calc_224()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA512_224_Add()	Add data to MAC.
CRYPTO_HMAC_SHA512_224_Init()	Initialize context.
CRYPTO_HMAC_SHA512_224_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA512_224_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA512_224_Final_224()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA512_224_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA512_224_Reset()	Reset MAC to initial state.

6.24.2.1 CRYPTO_HMAC_SHA512_224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Add( CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf,  
                                    const U8 * pInput,  
                                    unsigned InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/224 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.24.2.2 CRYPTO_HMAC_SHA512_224_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.24.2.3 CRYPTO_HMAC_SHA512_224_Calc_224()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Calc_224(      U8      * pOutput,
                                              const U8      * pKey,
                                              unsigned      KeyLen,
                                              const U8      * pInput,
                                              unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 28 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.24.2.4 CRYPTO_HMAC_SHA512_224_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Final(CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf,  
                                     U8                           * pOutput,  
                                     unsigned                      OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/224 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.24.2.5 CRYPTO_HMAC_SHA512_224_Final_224()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Final_224(CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf,  
                                         U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/224 context.
pOutput	Pointer to object that receives the MAC, 28 octets.

6.24.2.6 CRYPTO_HMAC_SHA512_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Init(      CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/224 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.24.2.7 CRYPTO_HMAC_SHA512_224_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA512_224_InitEx( CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf,
                                         const U8 * pKey,
                                         unsigned KeyLen,
                                         const U8 * pIV,
                                         unsigned IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.24.2.8 CRYPTO_HMAC_SHA512_224_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Kill(CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.24.2.9 CRYPTO_HMAC_SHA512_224_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA512_224_Reset(CRYPTO_HMAC_SHA512_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA512_224 context.

6.24.3 Generic API

The following table lists the HMAC-SHA-512/224 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA512_224_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA512_224_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA512_224_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA512_224_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA512_224_Final_224()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA512_224_Kill()</code>	Destroy MAC context.

6.24.3.1 CRYPTO_MAC_HMAC_SHA512_224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_Add(      void      * pContext,
                                             const U8      * pInput,
                                             unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.24.3.2 CRYPTO_MAC_HMAC_SHA512_224_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_Final(void      * pContext,
                                         U8        * pMAC,
                                         unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.24.3.3 CRYPTO_MAC_HMAC_SHA512_224_Final_224()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_Final_224(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.24.3.4 CRYPTO_MAC_HMAC_SHA512_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_Init(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.24.3.5 CRYPTO_MAC_HMAC_SHA512_224_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_InitEx(      void      * pContext,
                                                unsigned   DigestLen,
                                                const U8    * pKey,
                                                unsigned   KeyLen,
                                                const U8    * pIV,
                                                unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.24.3.6 CRYPTO_MAC_HMAC_SHA512_224_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.25 HMAC-SHA-512/256

6.25.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA-512/256 is specified by the following document:

- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*

6.25.2 Type-safe API

The following table lists the HMAC-SHA-512/256 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA512_256_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA512_256_Calc_256()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA512_256_Add()	Add data to MAC.
CRYPTO_HMAC_SHA512_256_Init()	Initialize context.
CRYPTO_HMAC_SHA512_256_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA512_256_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA512_256_Final_256()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA512_256_Reset()	Reset MAC to initial state.

6.25.2.1 CRYPTO_HMAC_SHA512_256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Add(          CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to HMAC-SHA-512/256 context.
<code>pInput</code>	Pointer to input octet string to add.
<code>InputLen</code>	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.25.2.2 CRYPTO_HMAC_SHA512_256_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.25.2.3 CRYPTO_HMAC_SHA512_256_Calc_256()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Calc_256(      U8      * pOutput,
                                              const U8      * pKey,
                                              unsigned      KeyLen,
                                              const U8      * pInput,
                                              unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 32 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.25.2.4 CRYPTO_HMAC_SHA512_256_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Final(CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf,  
                                     U8                           * pOutput,  
                                     unsigned                      OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/256 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.25.2.5 CRYPTO_HMAC_SHA512_256_Final_256()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Final_256(CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf,  
                                         U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/256 context.
pOutput	Pointer to object that receives the MAC, 32 octets.

6.25.2.6 CRYPTO_HMAC_SHA512_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Init(      CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA-512/256 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.25.2.7 CRYPTO_HMAC_SHA512_256_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA512_256_InitEx( CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf,  
                                     const U8 * pKey,  
                                     unsigned KeyLen,  
                                     const U8 * pIV,  
                                     unsigned IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.25.2.8 CRYPTO_HMAC_SHA512_256_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Kill(CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.25.2.9 CRYPTO_HMAC_SHA512_256_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA512_256_Reset(CRYPTO_HMAC_SHA512_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA512_256 context.

6.25.3 Generic API

The following table lists the HMAC-SHA-512/256 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA512_256_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA512_256_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SHA512_256_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA512_256_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA512_256_Final_256()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA512_256_Kill()</code>	Destroy MAC context.

6.25.3.1 CRYPTO_MAC_HMAC_SHA512_256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_Add(      void      * pContext,
                                             const U8      * pInput,
                                             unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.25.3.2 CRYPTO_MAC_HMAC_SHA512_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_Init(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.25.3.3 CRYPTO_MAC_HMAC_SHA512_256_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_InitEx(      void      * pContext,
                                                unsigned   DigestLen,
                                                const U8    * pKey,
                                                unsigned   KeyLen,
                                                const U8    * pIV,
                                                unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.25.3.4 CRYPTO_MAC_HMAC_SHA512_256_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_Final(void      * pContext,
                                         U8        * pMAC,
                                         unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.25.3.5 CRYPTO_MAC_HMAC_SHA512_256_Final_256()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_Final_256(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.25.3.6 CRYPTO_MAC_HMAC_SHA512_256_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA512_256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.26 HMAC-SHA3-224

6.26.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA3-224 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

6.26.2 Type-safe API

The following table lists the HMAC-SHA3-224 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA3_224_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA3_224_Calc_224()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA3_224_Init()	Initialize context.
CRYPTO_HMAC_SHA3_224_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA3_224_Add()	Add data to MAC.
CRYPTO_HMAC_SHA3_224_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA3_224_Final_224()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA3_224_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA3_224_Reset()	Reset MAC to initial state.

6.26.2.1 CRYPTO_HMAC_SHA3_224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Add(          CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-224 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.26.2.2 CRYPTO_HMAC_SHA3_224_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.26.2.3 CRYPTO_HMAC_SHA3_224_Calc_224()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Calc_224(      U8      * pOutput,
                                             const U8      * pKey,
                                             unsigned      KeyLen,
                                             const U8      * pInput,
                                             unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 28 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.26.2.4 CRYPTO_HMAC_SHA3_224_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Final(CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-224 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.26.2.5 CRYPTO_HMAC_SHA3_224_Final_224()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Final_224(CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf,  
                                      U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-224 context.
pOutput	Pointer to object that receives the MAC, 28 octets.

6.26.2.6 CRYPTO_HMAC_SHA3_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Init(          CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-224 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.26.2.7 CRYPTO_HMAC_SHA3_224_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA3_224_InitEx(      CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.26.2.8 CRYPTO_HMAC_SHA3_224_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Kill(CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.26.2.9 CRYPTO_HMAC_SHA3_224_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA3_224_Reset(CRYPTO_HMAC_SHA3_224_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3_224 context.

6.26.3 Generic API

The following table lists the HMAC-SHA3-224 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA3_224_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA3_224_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA3_224_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA3_224_Final_224()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA3_224_Kill()</code>	Destroy MAC context.

6.26.3.1 CRYPTO_MAC_HMAC_SHA3_224_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_224_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.26.3.2 CRYPTO_MAC_HMAC_SHA3_224_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_224_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.26.3.3 CRYPTO_MAC_HMAC_SHA3_224_Final_224()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_224_Final_224(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.26.3.4 CRYPTO_MAC_HMAC_SHA3_224_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_224_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.26.3.5 CRYPTO_MAC_HMAC_SHA3_224_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_224_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.27 HMAC-SHA3-256

6.27.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA3-256 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

6.27.2 Type-safe API

The following table lists the HMAC-SHA3-256 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA3_256_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA3_256_Calc_256()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA3_256_Init()	Initialize context.
CRYPTO_HMAC_SHA3_256_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA3_256_Add()	Add data to MAC.
CRYPTO_HMAC_SHA3_256_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA3_256_Final_256()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA3_256_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA3_256_Reset()	Reset MAC to initial state.

6.27.2.1 CRYPTO_HMAC_SHA3_256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Add(          CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-256 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.27.2.2 CRYPTO_HMAC_SHA3_256_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.27.2.3 CRYPTO_HMAC_SHA3_256_Calc_256()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Calc_256(      U8      * pOutput,
                                             const U8      * pKey,
                                             unsigned      KeyLen,
                                             const U8      * pInput,
                                             unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 32 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.27.2.4 CRYPTO_HMAC_SHA3_256_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Final(CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf,  
                                  U8                         * pOutput,  
                                  unsigned                     OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-256 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.27.2.5 CRYPTO_HMAC_SHA3_256_Final_256()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Final_256(CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-256 context.
pOutput	Pointer to object that receives the MAC, 32 octets.

6.27.2.6 CRYPTO_HMAC_SHA3_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Init(          CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-256 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.27.2.7 CRYPTO_HMAC_SHA3_256_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA3_256_InitEx(      CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.27.2.8 CRYPTO_HMAC_SHA3_256_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Kill(CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.27.2.9 CRYPTO_HMAC_SHA3_256_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA3_256_Reset(CRYPTO_HMAC_SHA3_256_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3_256 context.

6.27.3 Generic API

The following table lists the HMAC-SHA3-256 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA3_256_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA3_256_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA3_256_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA3_256_Final_256()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA3_256_Kill()</code>	Destroy MAC context.

6.27.3.1 CRYPTO_MAC_HMAC_SHA3_256_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_256_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.27.3.2 CRYPTO_MAC_HMAC_SHA3_256_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_256_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.27.3.3 CRYPTO_MAC_HMAC_SHA3_256_Final_256()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_256_Final_256(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.27.3.4 CRYPTO_MAC_HMAC_SHA3_256_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_256_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.27.3.5 CRYPTO_MAC_HMAC_SHA3_256_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_256_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.28 HMAC-SHA3-384

6.28.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA3-384 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

6.28.2 Type-safe API

The following table lists the HMAC-SHA3-384 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA3_384_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA3_384_Calc_384()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA3_384_Add()	Add data to MAC.
CRYPTO_HMAC_SHA3_384_Init()	Initialize context.
CRYPTO_HMAC_SHA3_384_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SHA3_384_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA3_384_Final_384()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA3_384_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA3_384_Reset()	Reset MAC to initial state.

6.28.2.1 CRYPTO_HMAC_SHA3_384_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Add(          CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-384 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.28.2.2 CRYPTO_HMAC_SHA3_384_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.28.2.3 CRYPTO_HMAC_SHA3_384_Calc_384()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Calc_384(      U8      * pOutput,
                                             const U8      * pKey,
                                             unsigned      KeyLen,
                                             const U8      * pInput,
                                             unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 48 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.28.2.4 CRYPTO_HMAC_SHA3_384_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Final(CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-384 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.28.2.5 CRYPTO_HMAC_SHA3_384_Final_384()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Final_384(CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf,  
                                      U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-384 context.
pOutput	Pointer to object that receives the MAC, 48 octets.

6.28.2.6 CRYPTO_HMAC_SHA3_384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Init(          CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-384 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.28.2.7 CRYPTO_HMAC_SHA3_384_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SHA3_384_InitEx(      CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen,
                                         const U8                      * pIV,
                                         unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.28.2.8 CRYPTO_HMAC_SHA3_384_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Kill(CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.28.2.9 CRYPTO_HMAC_SHA3_384_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA3_384_Reset(CRYPTO_HMAC_SHA3_384_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3_384 context.

6.28.3 Generic API

The following table lists the HMAC-SHA3-384 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA3_384_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA3_384_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA3_384_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA3_384_Final_384()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA3_384_Kill()</code>	Destroy MAC context.

6.28.3.1 CRYPTO_MAC_HMAC_SHA3_384_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_384_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.28.3.2 CRYPTO_MAC_HMAC_SHA3_384_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_384_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.28.3.3 CRYPTO_MAC_HMAC_SHA3_384_Final_384()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_384_Final_384(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.28.3.4 CRYPTO_MAC_HMAC_SHA3_384_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_384_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.28.3.5 CRYPTO_MAC_HMAC_SHA3_384_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_384_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.29 HMAC-SHA3-512

6.29.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SHA3-512 is specified by the following document:

- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

6.29.2 Type-safe API

The following table lists the HMAC-SHA3-512 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SHA3_512_Calc()	Calculate MAC.
CRYPTO_HMAC_SHA3_512_Calc_512()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SHA3_512_Add()	Add data to MAC.
CRYPTO_HMAC_SHA3_512_Init()	Initialize context.
CRYPTO_HMAC_SHA3_512_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SHA3_512_Final_512()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SHA3_512_Kill()	Destroy HMAC context.
CRYPTO_HMAC_SHA3_512_Reset()	Reset MAC to initial state.

6.29.2.1 CRYPTO_HMAC_SHA3_512_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Add(          CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-512 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.29.2.2 CRYPTO_HMAC_SHA3_512_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.29.2.3 CRYPTO_HMAC_SHA3_512_Calc_512()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Calc_512(      U8      * pOutput,
                                             const U8      * pKey,
                                             unsigned      KeyLen,
                                             const U8      * pInput,
                                             unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 64 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.29.2.4 CRYPTO_HMAC_SHA3_512_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Final(CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-512 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.29.2.5 CRYPTO_HMAC_SHA3_512_Final_512()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Final_512(CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-512 context.
pOutput	Pointer to object that receives the MAC, 64 octets.

6.29.2.6 CRYPTO_HMAC_SHA3_512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Init(          CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3-512 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.29.2.7 CRYPTO_HMAC_SHA3_512_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Kill(CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.29.2.8 CRYPTO_HMAC_SHA3_512_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SHA3_512_Reset(CRYPTO_HMAC_SHA3_512_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SHA3_512 context.

6.29.3 Generic API

The following table lists the HMAC-SHA3-512 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SHA512_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SHA512_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SHA512_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SHA512_Final_512()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SHA512_Kill()</code>	Destroy MAC context.

6.29.3.1 CRYPTO_MAC_HMAC_SHA3_512_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_512_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.29.3.2 CRYPTO_MAC_HMAC_SHA3_512_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_512_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.29.3.3 CRYPTO_MAC_HMAC_SHA3_512_Final_512()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_512_Final_512(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.29.3.4 CRYPTO_MAC_HMAC_SHA3_512_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_512_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.29.3.5 CRYPTO_MAC_HMAC_SHA3_512_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SHA3_512_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.30 HMAC-SM3

6.30.1 Standards reference

HMAC is specified by the following document:

- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*

It has an associated IETF RFC:

- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*

SM3 is specified by the following document:

- draft-sca-cfrg-sm3-02 — *The SM3 Cryptographic Hash Function*

6.30.2 Type-safe API

The following table lists the HMAC-SM3 type-safe API functions.

Function	Description
Message functions	
CRYPTO_HMAC_SM3_Calc()	Calculate MAC.
CRYPTO_HMAC_SM3_Calc_256()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_HMAC_SM3_Init()	Initialize context.
CRYPTO_HMAC_SM3_InitEx()	Initialize context, include subkey.
CRYPTO_HMAC_SM3_Add()	Add data to MAC.
CRYPTO_HMAC_SM3_Final()	Finalize MAC calculation.
CRYPTO_HMAC_SM3_Final_256()	Finalize MAC calculation, fixed size.
CRYPTO_HMAC_SM3_Reset()	Reset MAC to initial state.
CRYPTO_HMAC_SM3_Kill()	Destroy HMAC context.

6.30.2.1 CRYPTO_HMAC_SM3_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_HMAC_SM3_Add(      CRYPTO_HMAC_SM3_CONTEXT * pSelf,
                                const U8             * pInput,
                                unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SM3 context.
pInput	Pointer to input octet string to add.
InputLen	Octet length of the input octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.30.2.2 CRYPTO_HMAC_SM3_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_HMAC_SM3_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
<code>pOutput</code>	Pointer to object that receives the MAC.
<code>OutputLen</code>	Octet length of the MAC.
<code>pKey</code>	Pointer to key octet string.
<code>KeyLen</code>	Octet length of the key octet string.
<code>pInput</code>	Pointer to input octet string.
<code>InputLen</code>	Octet length of the input octet string.

6.30.2.3 CRYPTO_HMAC_SM3_Calc_256()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_HMAC_SM3_Calc_256(      U8      * pOutput,
                                       const U8      * pKey,
                                       unsigned      KeyLen,
                                       const U8      * pInput,
                                       unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 32 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.30.2.4 CRYPTO_HMAC_SM3_Final()

Description

Finalize MAC calculation.

Prototype

```
void CRYPTO_HMAC_SM3_Final(CRYPTO_HMAC_SM3_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SM3 context.
pOutput	Pointer to object that receive the MAC.
OutputLen	Octet length of the MAC.

6.30.2.5 CRYPTO_HMAC_SM3_Final_256()

Description

Finalize MAC calculation, fixed size.

Prototype

```
void CRYPTO_HMAC_SM3_Final_256(CRYPTO_HMAC_SM3_CONTEXT * pSelf,  
                                U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SM3 context.
pOutput	Pointer to object that receives the MAC, 32 octets.

6.30.2.6 CRYPTO_HMAC_SM3_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_HMAC_SM3_Init(      CRYPTO_HMAC_SM3_CONTEXT * pSelf,
                                const U8             * pKey,
                                unsigned            KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SM3 context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

6.30.2.7 CRYPTO_HMAC_SM3_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_HMAC_SM3_InitEx(          CRYPTO_HMAC_SM3_CONTEXT * pSelf ,  
                                const U8                      * pKey ,  
                                unsigned                     KeyLen ,  
                                const U8                      * pIV ,  
                                unsigned                     IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pIV	Pointer to initialization vector (unused).
IVLen	Octet length of the initialization vector (unused).

Additional information

As the HMAC algorithm does not support subkeys, the initialization vector is accepted but otherwise ignored.

6.30.2.8 CRYPTO_HMAC_SM3_Reset()

Description

Reset MAC to initial state.

Prototype

```
void CRYPTO_HMAC_SM3_Reset(CRYPTO_HMAC_SM3_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to HMAC-SM3 context.

6.30.2.9 CRYPTO_HMAC_SM3_Kill()

Description

Destroy HMAC context.

Prototype

```
void CRYPTO_HMAC_SM3_Kill(CRYPTO_HMAC_SM3_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.30.3 Generic API

The following table lists the HMAC-SM3 functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_HMAC_SM3_Init()</code>	Initialize context.
<code>CRYPTO_MAC_HMAC_SM3_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_HMAC_SM3_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_HMAC_SM3_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_HMAC_SM3_Final_256()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_HMAC_SM3_Kill()</code>	Destroy MAC context.

6.30.3.1 CRYPTO_MAC_HMAC_SM3_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.30.3.2 CRYPTO_MAC_HMAC_SM3_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.30.3.3 CRYPTO_MAC_HMAC_SM3_Final_256()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_Final_256(void * pContext,  
                                     U8     * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.30.3.4 CRYPTO_MAC_HMAC_SM3_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_Init(      void      * pContext,
                                     const U8      * pKey,
                                     unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.30.3.5 CRYPTO_MAC_HMAC_SM3_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.30.3.6 CRYPTO_MAC_HMAC_SM3_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_HMAC_SM3_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.31 XCBC-AES

6.31.1 Standards reference

AES-XCBC-MAC is specified by the following document:

- IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*

AES is specified by the following document:

- FIPS PUB 197 — *Specification for the Advanced Encryption Standard (AES)*

6.31.2 Type-safe API

The following table lists the XCBC-AES type-safe API functions.

Function	Description
Message functions	
CRYPTO_XCBC_AES_Calc()	Calculate MAC.
CRYPTO_XCBC_AES_Calc_96()	Calculate MAC, fixed size, truncated.
CRYPTO_XCBC_AES_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_XCBC_AES_Init()	Initialize context.
CRYPTO_XCBC_AES_InitEx()	Initialize context, include subkey.
CRYPTO_XCBC_AES_Add()	Add data to MAC.
CRYPTO_XCBC_AES_Final()	Finish MAC calculation.
CRYPTO_XCBC_AES_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_XCBC_AES_Kill()	Destroy context.

6.31.2.1 CRYPTO_XCBC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_XCBC_AES_Add(      CRYPTO_XCBC_AES_CONTEXT * pSelf,
                               const U8           * pInput,
                               unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.31.2.2 CRYPTO_XCBC_AES_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_XCBC_AES_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.31.2.3 CRYPTO_XCBC_AES_Calc_96()

Description

Calculate MAC, fixed size, truncated.

Prototype

```
void CRYPTO_XCBC_AES_Calc_96(      U8      * pOutput,
                                     const U8      * pKey,
                                     unsigned      KeyLen,
                                     const U8      * pInput,
                                     unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 12 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.31.2.4 CRYPTO_XCBC_AES_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_XCBC_AES_Calc_128(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.31.2.5 CRYPTO_XCBC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_XCBC_AES_Final(CRYPTO_XCBC_AES_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.31.2.6 CRYPTO_XCBC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_XCBC_AES_Final_128(CRYPTO_XCBC_AES_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.31.2.7 CRYPTO_XCBC_AES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_XCBC_AES_Init(      CRYPTO_XCBC_AES_CONTEXT * pSelf,
                                const U8             * pKey,
                                unsigned            KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for AES-XCBC-MAC.

6.31.2.8 CRYPTO_XCBC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_XCBC_AES_InitEx(          CRYPTO_XCBC_AES_CONTEXT * pSelf ,  
                                const U8                      * pKey ,  
                                unsigned                     KeyLen ,  
                                const U8                      * pIV ,  
                                unsigned                     IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for AES-XCBC-MAC.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.31.2.9 CRYPTO_XCBC_AES_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_XCBC_AES_Kill(CRYPTO_XCBC_AES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.31.3 Generic API

The following table lists the XCBC-AES functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_XCBC_AES_Init()</code>	Initialize context.
<code>CRYPTO_MAC_XCBC_AES_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_XCBC_AES_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_XCBC_AES_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_XCBC_AES_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_XCBC_AES_Kill()</code>	Destroy MAC context.

6.31.3.1 CRYPTO_MAC_XCBC_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_XCBC_AES_Add(      void      * pContext,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.31.3.2 CRYPTO_MAC_XCBC_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_XCBC_AES_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.31.3.3 CRYPTO_MAC_XCBC_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_XCBC_AES_Final_128(void * pContext,  
                                     U8     * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.31.3.4 CRYPTO_MAC_XCBC_AES_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_XCBC_AES_Init(      void      * pContext,
                                     const U8      * pKey,
                                     unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.31.3.5 CRYPTO_MAC_XCBC_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_XCBC_AES_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.31.3.6 CRYPTO_MAC_XCBC_AES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_XCBC_AES_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.31.4 Self-test API

The following table lists the XCBC-AES self-test API functions.

Function	Description
CRYPTO_XCBC_AES_RFC3566_SelfTest()	Run SM3 KATs from GBT.

6.31.4.1 CRYPTO_XCBC_AES_RFC3566_SelfTest()

Description

Run SM3 KATs from GBT.

Prototype

```
void CRYPTO_XCBC_AES_RFC3566_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.32 XCBC-SEED

6.32.1 Standards reference

SEED-XCBC-MAC uses the AES-XCBC-MAC algorithm with SEED substituted for AES as the cipher.

AES-XCBC-MAC is specified by the following document:

- IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*

SEED is specified by the following document:

- IETF RFC 4269 — *The SEED Encryption Algorithm*

6.32.2 Type-safe API

The following table lists the Xcbc-Seed type-safe API functions.

Function	Description
Message functions	
CRYPTO_XCBC_SEED_Calc()	Calculate MAC.
CRYPTO_XCBC_SEED_Calc_96()	Calculate MAC, fixed size, truncated.
CRYPTO_XCBC_SEED_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_XCBC_SEED_Init()	Initialize context.
CRYPTO_XCBC_SEED_InitEx()	Initialize context, include subkey.
CRYPTO_XCBC_SEED_Add()	Add data to MAC.
CRYPTO_XCBC_SEED_Final()	Finish MAC calculation.
CRYPTO_XCBC_SEED_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_XCBC_SEED_Kill()	Destroy context.

6.32.2.1 CRYPTO_XCBC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_XCBC_SEED_Add(          CRYPTO_XCBC_SEED_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.32.2.2 CRYPTO_XCBC_SEED_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_XCBC_SEED_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
                                  unsigned KeyLen,
const U8      * pInput,
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.32.2.3 CRYPTO_XCBC_SEED_Calc_96()

Description

Calculate MAC, fixed size, truncated.

Prototype

```
void CRYPTO_XCBC_SEED_Calc_96(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 12 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.32.2.4 CRYPTO_XCBC_SEED_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_XCBC_SEED_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.32.2.5 CRYPTO_XCBC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_XCBC_SEED_Final(CRYPTO_XCBC_SEED_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.32.2.6 CRYPTO_XCBC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_XCBC_SEED_Final_128(CRYPTO_XCBC_SEED_CONTEXT * pSelf,  
                                  U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.32.2.7 CRYPTO_XCBC_SEED_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_XCBC_SEED_Init(      CRYPTO_XCBC_SEED_CONTEXT * pSelf,
                                const U8               * pKey,
                                unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for SEED-XCBC-MAC.

6.32.2.8 CRYPTO_XCBC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_XCBC_SEED_InitEx(      CRYPTO_XCBC_SEED_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for SEED-XCBC-MAC.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.32.2.9 CRYPTO_XCBC_SEED_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_XCBC_SEED_Kill(CRYPTO_XCBC_SEED_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.32.3 Generic API

The following table lists the XCBC-SEED functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_XCBC_SEED_Init()</code>	Initialize context.
<code>CRYPTO_MAC_XCBC_SEED_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_XCBC_SEED_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_XCBC_SEED_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_XCBC_SEED_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_XCBC_SEED_Kill()</code>	Destroy MAC context.

6.32.3.1 CRYPTO_MAC_XCBC_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.32.3.2 CRYPTO_MAC_XCBC_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_Final(void      * pContext,
                                U8        * pMAC,
                                unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.32.3.3 CRYPTO_MAC_XCBC_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.32.3.4 CRYPTO_MAC_XCBC_SEED_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.32.3.5 CRYPTO_MAC_XCBC_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.32.3.6 CRYPTO_MAC_XCBC_SEED_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_XCBC_SEED_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.33 XCBC-ARIA

6.33.1 Standards reference

ARIA-XCBC-MAC uses the AES-XCBC-MAC algorithm with ARIA substituted for AES as the cipher.

AES-XCBC-MAC is specified by the following document:

- IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*

ARIA is specified by the following document:

- IETF RFC 5794 — *A Description of the ARIA Encryption Algorithm*

6.33.2 Type-safe API

The following table lists the XCBC-ARIA type-safe API functions.

Function	Description
Message functions	
CRYPTO_XCBC_ARIA_Calc()	Calculate MAC.
CRYPTO_XCBC_ARIA_Calc_96()	Calculate MAC, fixed size, truncated.
CRYPTO_XCBC_ARIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_XCBC_ARIA_Init()	Initialize context.
CRYPTO_XCBC_ARIA_InitEx()	Initialize context, include subkey.
CRYPTO_XCBC_ARIA_Add()	Add data to MAC.
CRYPTO_XCBC_ARIA_Final()	Finish MAC calculation.
CRYPTO_XCBC_ARIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_XCBC_ARIA_Kill()	Destroy context.

6.33.2.1 CRYPTO_XCBC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_XCBC_ARIA_Add(          CRYPTO_XCBC_ARIA_CONTEXT * pSelf,
                                const U8                  * pInput,
                                unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.33.2.2 CRYPTO_XCBC_ARIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_XCBC_ARIA_Calc(      U8      * pOutput,
                                  unsigned OutputLen,
const U8      * pKey,
unsigned   KeyLen,
const U8      * pInput,
unsigned  InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.33.2.3 CRYPTO_XCBC_ARIA_Calc_96()

Description

Calculate MAC, fixed size, truncated.

Prototype

```
void CRYPTO_XCBC_ARIA_Calc_96(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 12 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.33.2.4 CRYPTO_XCBC_ARIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_XCBC_ARIA_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.33.2.5 CRYPTO_XCBC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_XCBC_ARIA_Final(CRYPTO_XCBC_ARIA_CONTEXT * pSelf,  
                           U8 * pOutput,  
                           unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.33.2.6 CRYPTO_XCBC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_XCBC_ARIA_Final_128(CRYPTO_XCBC_ARIA_CONTEXT * pSelf,  
                                  U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.33.2.7 CRYPTO_XCBC_ARIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_XCBC_ARIA_Init(      CRYPTO_XCBC_ARIA_CONTEXT * pSelf,
                                  const U8               * pKey,
                                  unsigned             KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for ARIA-XCBC-MAC.

6.33.2.8 CRYPTO_XCBC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_XCBC_ARIA_InitEx(      CRYPTO_XCBC_ARIA_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen,
                                     const U8                      * pIV,
                                     unsigned                     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for ARIA-XCBC-MAC.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.33.2.9 CRYPTO_XCBC_ARIA_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_XCBC_ARIA_Kill(CRYPTO_XCBC_ARIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.33.3 Generic API

The following table lists the XCBC-ARIA functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_XCBC_ARIA_Init()</code>	Initialize context.
<code>CRYPTO_MAC_XCBC_ARIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_XCBC_ARIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_XCBC_ARIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_XCBC_ARIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_XCBC_ARIA_Kill()</code>	Destroy MAC context.

6.33.3.1 CRYPTO_MAC_XCBC_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_Add(      void      * pContext,
                                     const U8      * pInput,
                                     unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.33.3.2 CRYPTO_MAC_XCBC_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_Final(void      * pContext,
                                  U8        * pMAC,
                                  unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.33.3.3 CRYPTO_MAC_XCBC_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_Final_128(void * pContext,  
                                     U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.33.3.4 CRYPTO_MAC_XCBC_ARIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.33.3.5 CRYPTO_MAC_XCBC_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8   * pKey,
                                         unsigned   KeyLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.33.3.6 CRYPTO_MAC_XCBC_ARIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_XCBC_ARIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.34 XCBC-Camellia

6.34.1 Standards reference

Camellia-XCBC-MAC uses the AES-XCBC-MAC algorithm with Camellia substituted for AES as the cipher.

AES-XCBC-MAC is specified by the following document:

- [IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*](#)

Camellia is specified by the following document:

- [IETF RFC 3713 — *A Description of the Camellia Encryption Algorithm*](#)

6.34.2 Type-safe API

The following table lists the XCBC-Camellia type-safe API functions.

Function	Description
Message functions	
CRYPTO_XCBC_CAMELLIA_Calc()	Calculate MAC.
CRYPTO_XCBC_CAMELLIA_Calc_96()	Calculate MAC, fixed size, truncated.
CRYPTO_XCBC_CAMELLIA_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_XCBC_CAMELLIA_Init()	Initialize context.
CRYPTO_XCBC_CAMELLIA_InitEx()	Initialize context, include subkey.
CRYPTO_XCBC_CAMELLIA_Add()	Add data to MAC.
CRYPTO_XCBC_CAMELLIA_Final()	Finish MAC calculation.
CRYPTO_XCBC_CAMELLIA_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_XCBC_CAMELLIA_Kill()	Destroy context.

6.34.2.1 CRYPTO_XCBC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Add(          CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.34.2.2 CRYPTO_XCBC_CAMELLIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Calc(      U8      * pOutput,
                                         unsigned   OutputLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.34.2.3 CRYPTO_XCBC_CAMELLIA_Calc_96()

Description

Calculate MAC, fixed size, truncated.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Calc_96(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 12 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.34.2.4 CRYPTO_XCBC_CAMELLIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Calc_128(      U8      * pOutput,
                                           const U8      * pKey,
                                           unsigned      KeyLen,
                                           const U8      * pInput,
                                           unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.34.2.5 CRYPTO_XCBC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Final(CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf,  
                                  U8  
                                  unsigned  
                                  * pOutput,  
                                  OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.34.2.6 CRYPTO_XCBC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Final_128(CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf,  
                                      U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.34.2.7 CRYPTO_XCBC_CAMELLIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Init(          CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf,
                                         const U8                      * pKey,
                                         unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for CAMELLIA-XCBC-MAC.

6.34.2.8 CRYPTO_XCBC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_InitEx(      CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf,
                                         const U8               * pKey,
                                         unsigned                KeyLen,
                                         const U8               * pIV,
                                         unsigned                IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for CAMELLIA-XCBC-MAC.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.34.2.9 CRYPTO_XCBC_CAMELLIA_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_XCBC_CAMELLIA_Kill(CRYPTO_XCBC_CAMELLIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.34.3 Generic API

The following table lists the XCBC-Camellia functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_XCBC_CAMELLIA_Init()</code>	Initialize context.
<code>CRYPTO_MAC_XCBC_CAMELLIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_XCBC_CAMELLIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_XCBC_CAMELLIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_XCBC_CAMELLIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_XCBC_CAMELLIA_Kill()</code>	Destroy MAC context.

6.34.3.1 CRYPTO_MAC_XCBC_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.34.3.2 CRYPTO_MAC_XCBC_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_Final(void      * pContext,
                                      U8        * pMAC,
                                      unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.34.3.3 CRYPTO_MAC_XCBC_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.34.3.4 CRYPTO_MAC_XCBC_CAMELLIA_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.34.3.5 CRYPTO_MAC_XCBC_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.34.3.6 CRYPTO_MAC_XCBC_CAMELLIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_XCBC_CAMELLIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.35 XCBC-Twofish

6.35.1 Standards reference

Twofish-XCBC-MAC uses the AES-XCBC-MAC algorithm with Twofish substituted for AES as the cipher.

AES-XCBC-MAC is specified by the following document:

- [IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*](#)

Twofish is specified by the following document:

- [Schneier at al — *Twofish: A 128-Bit Block Cipher*](#)

6.35.2 Type-safe API

The following table lists the XCBC-Twofish type-safe API functions.

Function	Description
Message functions	
CRYPTO_XCBC_TWOFISH_Calc()	Calculate MAC.
CRYPTO_XCBC_TWOFISH_Calc_96()	Calculate MAC, fixed size, truncated.
CRYPTO_XCBC_TWOFISH_Calc_128()	Calculate MAC, fixed size.
Incremental functions	
CRYPTO_XCBC_TWOFISH_Init()	Initialize context.
CRYPTO_XCBC_TWOFISH_InitEx()	Initialize context, include subkey.
CRYPTO_XCBC_TWOFISH_Add()	Add data to MAC.
CRYPTO_XCBC_TWOFISH_Final()	Finish MAC calculation.
CRYPTO_XCBC_TWOFISH_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_XCBC_TWOFISH_Kill()	Destroy context.

6.35.2.1 CRYPTO_XCBC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Add(          CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.35.2.2 CRYPTO_XCBC_TWOFISH_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Calc(      U8      * pOutput,
                                     unsigned OutputLen,
                                     const U8   * pKey,
                                     unsigned KeyLen,
                                     const U8   * pInput,
                                     unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
OutputLen	Octet length of the MAC.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.35.2.3 CRYPTO_XCBC_TWOFISH_Calc_96()

Description

Calculate MAC, fixed size, truncated.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Calc_96(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pInput,
                                         unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 12 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.35.2.4 CRYPTO_XCBC_TWOFISH_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Calc_128(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.35.2.5 CRYPTO_XCBC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Final(CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf,  
                                U8  
                                unsigned  
                                * pOutput,  
                                OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.35.2.6 CRYPTO_XCBC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Final_128(CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf,  
                                     U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.35.2.7 CRYPTO_XCBC_TWOFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Init(      CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf,
                                     const U8                      * pKey,
                                     unsigned                     KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for TWOFISH-XCBC-MAC.

6.35.2.8 CRYPTO_XCBC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_XCBC_TWOFISH_InitEx(          CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf ,  
                                     const U8                      * pKey ,  
                                     unsigned                     KeyLen ,  
                                     const U8                      * pIV ,  
                                     unsigned                     IVLen );
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key, fixed at 16 for TWOFISH-XCBC-MAC.
pIV	Pointer to initialization vector (ignored).
IVLen	Octet length of the initialization vector (must be zero).

6.35.2.9 CRYPTO_XCBC_TWOFISH_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_XCBC_TWOFISH_Kill(CRYPTO_XCBC_TWOFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.35.3 Generic API

The following table lists the XCBC-Twofish functions that conform to the generic MAC API.

Function	Description
CRYPTO_MAC_XCBC_TWOFISH_Init()	Initialize context.
CRYPTO_MAC_XCBC_TWOFISH_InitEx()	Initialize context, include subkey.
CRYPTO_MAC_XCBC_TWOFISH_Add()	Add data to MAC.
CRYPTO_MAC_XCBC_TWOFISH_Final()	Finish MAC calculation.
CRYPTO_MAC_XCBC_TWOFISH_Final_128()	Finish MAC calculation, fixed size.
CRYPTO_MAC_XCBC_TWOFISH_Kill()	Destroy MAC context.

6.35.3.1 CRYPTO_MAC_XCBC_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.35.3.2 CRYPTO_MAC_XCBC_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_Final(void * pContext,  
                                    U8      * pMAC,  
                                    unsigned MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.35.3.3 CRYPTO_MAC_XCBC_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.35.3.4 CRYPTO_MAC_XCBC_TWOFISH_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_Init(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.35.3.5 CRYPTO_MAC_XCBC_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.35.3.6 CRYPTO_MAC_XCBC_TWOFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_XCBC_TWOFISH_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.36 KMAC

6.36.1 Standards reference

KMAC is specified by the following document:

- NIST SP 800-185 — *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*

6.36.2 Type-safe API

The following table lists the KMAC type-safe API functions.

Function	Description
Incremental functions	
CRYPTO_KMAC_Init()	Initialize KMAC.
CRYPTO_KMAC_128_Init()	Initialize KMAC128.
CRYPTO_KMAC_256_Init()	Initialize KMAC256.
CRYPTO_KMAC_Add()	Add data to MAC.
CRYPTO_KMAC_Get()	Get KMAC.

6.36.2.1 CRYPTO_KMAC_Init()

Description

Initialize KMAC.

Prototype

```
void CRYPTO_KMAC_Init(      CRYPTO_KMAC_CONTEXT * pSelf,
                           const U8           * pKey,
                           unsigned           KeyLen,
                           const U8           * pCust,
                           unsigned           CustLen,
                           unsigned           Security);
```

Parameters

Parameter	Description
pSelf	Pointer to KMAC context.
pKey	Pointer to key string.
KeyLen	Octet length of the key string.
pCust	Pointer to customization string, S.
CustLen	Octet length of the customization string.
Security	Security strength in bits.

6.36.2.2 CRYPTO_KMAC_128_Init()

Description

Initialize KMAC128.

Prototype

```
void CRYPTO_KMAC_128_Init(      CRYPTO_KMAC_CONTEXT * pSelf,
                                const U8          * pKey,
                                unsigned           KeyLen,
                                const U8          * pCust,
                                unsigned           CustLen);
```

Parameters

Parameter	Description
pSelf	Pointer to KMAC context.
pKey	Pointer to key string.
KeyLen	Octet length of the key string.
pCust	Pointer to customization string, S.
CustLen	Octet length of the customization string.

6.36.2.3 CRYPTO_KMAC_256_Init()

Description

Initialize KMAC256.

Prototype

```
void CRYPTO_KMAC_256_Init(      CRYPTO_KMAC_CONTEXT * pSelf,
                                const U8          * pKey,
                                unsigned           KeyLen,
                                const U8          * pCust,
                                unsigned           CustLen);
```

Parameters

Parameter	Description
pSelf	Pointer to KMAC context.
pKey	Pointer to key string.
KeyLen	Octet length of the key string.
pCust	Pointer to customization string, S.
CustLen	Octet length of the customization string.

6.36.2.4 CRYPTO_KMAC_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_KMAC_Add(      CRYPTO_KMAC_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to KMAC context.
pInput	Pointer to input string to add to MAC.
InputLen	Octet length of the input string.

6.36.2.5 CRYPTO_KMAC_Get()

Description

Get KMAC.

Prototype

```
void CRYPTO_KMAC_Get(CRYPTO_KMAC_CONTEXT * pSelf,  
                      U8                  * pMAC,  
                      unsigned           MACLen);
```

Parameters

Parameter	Description
pSelf	Pointer to KMAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the requested MAC.

6.36.3 Self-test API

The following table lists the KMAC self-test API functions.

Function	Description
CRYPTO_KMAC_CSRC_SelfTest()	Run all CSRC KMAC validation tests.

6.36.3.1 CRYPTO_KMAC_CSRC_SelfTest()

Description

Run all CSRC KMAC validation tests.

Prototype

```
void CRYPTO_KMAC_CSRC_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.37 Poly1305

6.37.1 Type-safe API

The following table lists the Poly1305 type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_POLY1305_Calc()</code>	Calculate MAC.
<code>CRYPTO_POLY1305_Calc_128()</code>	Calculate MAC, fixed size.
Incremental functions	
<code>CRYPTO_POLY1305_Init()</code>	Initialize MAC.
<code>CRYPTO_POLY1305_Init_256()</code>	Initialize MAC, 256-bit key.
<code>CRYPTO_POLY1305_Add()</code>	Add data to MAC.
<code>CRYPTO_POLY1305_Final()</code>	Finalize MAC.
<code>CRYPTO_POLY1305_Final_128()</code>	Finalize MAC, fixed size.
<code>CRYPTO_POLY1305_Kill()</code>	Destroy MAC.
<code>CRYPTO_POLY1305_Clamp()</code>	Clamp key.

6.37.1.1 CRYPTO_POLY1305_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_POLY1305_Add(      CRYPTO_POLY1305_CONTEXT * pSelf,
                                const U8             * pInput,
                                unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.
pInput	Pointer to input string to add to MAC.
InputLen	Octet length of the input string.

6.37.1.2 CRYPTO_POLY1305_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_Calc(      U8      * pOutput,
                                unsigned OutputLen,
                                const U8      * pKey,
                                unsigned KeyLen,
                                const U8      * pInput,
                                unsigned InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.37.1.3 CRYPTO_POLY1305_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_Calc_128(      U8      * pOutput,
                                      const U8      * pKey,
                                      unsigned      KeyLen,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 16 octets.
pKey	Pointer to key.
KeyLen	Octet length of the key.
pInput	Pointer to message.
InputLen	Octet length of the message.

6.37.1.4 CRYPTO_POLY1305_Final()

Description

Finalize MAC.

Prototype

```
void CRYPTO_POLY1305_Final(CRYPTO_POLY1305_CONTEXT * pSelf,  
                           U8                      * pMAC,  
                           unsigned                 MACLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the requested MAC.

6.37.1.5 CRYPTO_POLY1305_Final_128()

Description

Finalize MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_Final_128(CRYPTO_POLY1305_CONTEXT * pSelf,  
                                U8 * pMAC);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.
pMAC	Pointer to object that receives the MAC, 16 octets.

6.37.1.6 CRYPTO_POLY1305_Init()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_Init(      CRYPTO_POLY1305_CONTEXT * pSelf,
                                const U8             * pKey,
                                unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.
pKey	Pointer to key string.
KeyLen	Octet length of the key string, must be 32.

6.37.1.7 CRYPTO_POLY1305_Init_256()

Description

Initialize MAC, 256-bit key.

Prototype

```
void CRYPTO_POLY1305_Init_256(      CRYPTO_POLY1305_CONTEXT * pSelf,
                                         const U8                      * pKey);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.
pKey	Pointer to key string, 32 octets.

6.37.1.8 CRYPTO_POLY1305_Kill()

Description

Destroy MAC.

Prototype

```
void CRYPTO_POLY1305_Kill(CRYPTO_POLY1305_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305 context.

6.37.1.9 CRYPTO_POLY1305_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

Additional information

The Poly1305 key "rs" is the concatenation of two 16-byte octet strings, r and s, where the initial 16-byte octet string s must be modified to clear a proportion of bits before use.

Key octets with indexes 3, 7, 11, and 15 are required to have their top four bits clear and octets with indexes 4, 8, and 12 are required to have their bottom two bits clear.

6.37.2 Self-test API

The following table lists the Poly1305 self-test API functions.

Function	Description
CRYPTO_POLY1305_Bernstein_SelfTest()	Run Poly1305 KAT from Bernstein's NaCl.

6.37.2.1 CRYPTO_POLY1305_Bernstein_SelfTest()

Description

Run Poly1305 KAT from Bernstein's NaCl.

Prototype

```
void CRYPTO_POLY1305_Bernstein_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.38 Poly1305-AES

6.38.1 Type-safe API

The following table lists the Poly1305-AES type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_POLY1305_AES_Calc()</code>	Calculate MAC.
<code>CRYPTO_POLY1305_AES_Calc_128()</code>	Calculate MAC, fixed size.
<code>CRYPTO_POLY1305_AES_Verify()</code>	Verify MAC.
<code>CRYPTO_POLY1305_AES_Verify_128()</code>	Verify MAC, fixed size.
Incremental functions	
<code>CRYPTO_POLY1305_AES_InitEx_256_128()</code>	Initialize MAC.
<code>CRYPTO_POLY1305_AES_Add()</code>	Add to MAC.
<code>CRYPTO_POLY1305_AES_Final()</code>	Compute MAC.
<code>CRYPTO_POLY1305_AES_Final_128()</code>	Compute MAC, fixed size.
<code>CRYPTO_POLY1305_AES_Kill()</code>	Destroy MAC context.
Key functions	
<code>CRYPTO_POLY1305_AES_Clamp()</code>	Clamp key.

6.38.1.1 CRYPTO_POLY1305_AES_Add()

Description

Add to MAC.

Prototype

```
void CRYPTO_POLY1305_AES_Add(          CRYPTO_POLY1305_AES_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.38.1.2 CRYPTO_POLY1305_AES_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_AES_Calc(      U8      * pTag,
                                         unsigned   TagLen,
                                         const U8    * pKey,
                                         const U8    * pIV,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag.
TagLen	Octet length of the requested authentication tag, at most 16 octets.
pKey	Pointer to key octet string, 32 octets.
pIV	Pointer to IV octet string, 16 octets.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.38.1.3 CRYPTO_POLY1305_AES_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_AES_Calc_128(      U8      * pTag,
                                         const U8      * pKey,
                                         const U8      * pIV,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.38.1.4 CRYPTO_POLY1305_AES_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_AES_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

6.38.1.5 CRYPTO_POLY1305_AES_Final()

Description

Compute MAC.

Prototype

```
void CRYPTO_POLY1305_AES_Final(CRYPTO_POLY1305_AES_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-AES context.
pOutput	Pointer to object that receives the authentication tag.
OutputLen	Octet length of the requested authentication tag.

6.38.1.6 CRYPTO_POLY1305_AES_Final_128()

Description

Compute MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_AES_Final_128(CRYPTO_POLY1305_AES_CONTEXT * pSelf,  
                                     U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-AES context.
pOutput	Pointer to object that receives the authentication tag, 16 octets.

6.38.1.7 CRYPTO_POLY1305_AES_InitEx_256_128()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_AES_InitEx_256_128( CRYPTO_POLY1305_AES_CONTEXT * pSelf ,  
                                         const U8 * pKey ,  
                                         const U8 * pIV );
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-AES context.
pKey	Pointer to key octet string, 32 bytes.
pIV	Pointer to IV octet string, 16 bytes.

6.38.1.8 CRYPTO_POLY1305_AES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_POLY1305_AES_Kill(CRYPTO_POLY1305_AES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-AES context.

6.38.1.9 CRYPTO_POLY1305_AES_Verify()

Description

Verify MAC.

Prototype

```
int CRYPTO_POLY1305_AES_Verify(const U8      * pTag,
                                unsigned     TagLen,
                                const U8      * pKey,
                                const U8      * pIV,
                                const U8      * pInput,
                                unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag.
TagLen	Octet length of the authentication tag.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.38.1.10 CRYPTO_POLY1305_AES_Verify_128()

Description

Verify MAC, fixed size.

Prototype

```
int CRYPTO_POLY1305_AES_Verify_128(const U8      * pTag,
                                      const U8      * pKey,
                                      const U8      * pIV,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.38.2 Generic API

The following table lists the Poly1305-AES functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_POLY1305_AES_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_POLY1305_AES_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_POLY1305_AES_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_POLY1305_AES_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_POLY1305_AES_Kill()</code>	Destroy MAC context.

6.38.2.1 CRYPTO_MAC_POLY1305_AES_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_POLY1305_AES_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.38.2.2 CRYPTO_MAC_POLY1305_AES_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_POLY1305_AES_Final(void      * pContext,
                                     U8        * pMAC,
                                     unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.38.2.3 CRYPTO_MAC_POLY1305_AES_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_POLY1305_AES_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.38.2.4 CRYPTO_MAC_POLY1305_AES_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_POLY1305_AES_InitEx(      void      * pContext,
                                             unsigned   DigestLen,
                                             const U8    * pKey,
                                             unsigned   KeyLen,
                                             const U8    * pIV,
                                             unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.38.2.5 CRYPTO_MAC_POLY1305_AES_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_POLY1305_AES_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.38.3 Self-test API

The following table lists the Poly1305 self-test API functions.

Function	Description
CRYPTO_POLY1305_AES_Bernstein_SelfTest	Run Poly1305-AES KATs from Bernstein.

6.38.3.1 CRYPTO_POLY1305_AES_Bernstein_SelfTest()

Description

Run Poly1305-AES KATs from Bernstein.

Prototype

```
void CRYPTO_POLY1305_AES_Bernstein_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

6.39 Poly1305-SEED

6.39.1 Type-safe API

The following table lists the Poly1305-SEED type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_POLY1305_SEED_Calc()</code>	Calculate MAC.
<code>CRYPTO_POLY1305_SEED_Calc_128()</code>	Calculate MAC, fixed size.
<code>CRYPTO_POLY1305_SEED_Verify()</code>	Verify MAC.
<code>CRYPTO_POLY1305_SEED_Verify_128()</code>	Verify MAC, fixed size.
Incremental functions	
<code>CRYPTO_POLY1305_SEED_InitEx_256_128()</code>	Initialize MAC.
<code>CRYPTO_POLY1305_SEED_Add()</code>	Add to MAC.
<code>CRYPTO_POLY1305_SEED_Final()</code>	Compute MAC.
<code>CRYPTO_POLY1305_SEED_Final_128()</code>	Compute MAC, fixed size.
<code>CRYPTO_POLY1305_SEED_Kill()</code>	Destroy MAC context.
Key functions	
<code>CRYPTO_POLY1305_SEED_Clamp()</code>	Clamp key.

6.39.1.1 CRYPTO_POLY1305_SEED_Add()

Description

Add to MAC.

Prototype

```
void CRYPTO_POLY1305_SEED_Add(      CRYPTO_POLY1305_SEED_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.39.1.2 CRYPTO_POLY1305_SEED_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_SEED_Calc(      U8      * pTag,
                                         unsigned   TagLen,
                                         const U8    * pKey,
                                         const U8    * pIV,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag.
TagLen	Octet length of the requested authentication tag, at most 16 octets.
pKey	Pointer to key octet string, 32 octets.
pIV	Pointer to IV octet string, 16 octets.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.39.1.3 CRYPTO_POLY1305_SEED_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_SEED_Calc_128(      U8      * pTag,
                                         const U8      * pKey,
                                         const U8      * pIV,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.39.1.4 CRYPTO_POLY1305_SEED_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_SEED_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

6.39.1.5 CRYPTO_POLY1305_SEED_Final()

Description

Compute MAC.

Prototype

```
void CRYPTO_POLY1305_SEED_Final(CRYPTO_POLY1305_SEED_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-SEED context.
pOutput	Pointer to object that receives the authentication tag.
OutputLen	Octet length of the requested authentication tag.

6.39.1.6 CRYPTO_POLY1305_SEED_Final_128()

Description

Compute MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_SEED_Final_128(CRYPTO_POLY1305_SEED_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-SEED context.
pOutput	Pointer to object that receives the authentication tag, 16 octets.

6.39.1.7 CRYPTO_POLY1305_SEED_InitEx_256_128()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_SEED_InitEx_256_128
(
    CRYPTO_POLY1305_SEED_CONTEXT * pSelf,
    const U8                      * pKey,
    const U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-SEED context.
pKey	Pointer to key octet string, 32 bytes.
pIV	Pointer to IV octet string, 16 bytes.

6.39.1.8 CRYPTO_POLY1305_SEED_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_POLY1305_SEED_Kill(CRYPTO_POLY1305_SEED_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to Poly1305-SEED context.

6.39.1.9 CRYPTO_POLY1305_SEED_Verify()

Description

Verify MAC.

Prototype

```
int CRYPTO_POLY1305_SEED_Verify(const U8      * pTag,
                                  unsigned     TagLen,
                                  const U8      * pKey,
                                  const U8      * pIV,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag.
TagLen	Octet length of the authentication tag.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.39.1.10 CRYPTO_POLY1305_SEED_Verify_128()

Description

Verify MAC, fixed size.

Prototype

```
int CRYPTO_POLY1305_SEED_Verify_128(const U8      * pTag,
                                      const U8      * pKey,
                                      const U8      * pIV,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.39.2 Generic API

The following table lists the Poly1305-SEED functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_POLY1305_SEED_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_POLY1305_SEED_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_POLY1305_SEED_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_POLY1305_SEED_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_POLY1305_SEED_Kill()</code>	Destroy MAC context.

6.39.2.1 CRYPTO_MAC_POLY1305_SEED_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_POLY1305_SEED_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.39.2.2 CRYPTO_MAC_POLY1305_SEED_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_POLY1305_SEED_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.39.2.3 CRYPTO_MAC_POLY1305_SEED_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_POLY1305_SEED_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.39.2.4 CRYPTO_MAC_POLY1305_SEED_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_POLY1305_SEED_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.39.2.5 CRYPTO_MAC_POLY1305_SEED_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_POLY1305_SEED_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.40 Poly1305-ARIA

6.40.1 Type-safe API

The following table lists the Poly1305-ARIA type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_POLY1305_ARIA_Calc()</code>	Calculate MAC.
<code>CRYPTO_POLY1305_ARIA_Calc_128()</code>	Calculate MAC, fixed size.
<code>CRYPTO_POLY1305_ARIA_Verify()</code>	Verify MAC.
<code>CRYPTO_POLY1305_ARIA_Verify_128()</code>	Verify MAC, fixed size.
Incremental functions	
<code>CRYPTO_POLY1305_ARIA_InitEx_256_128()</code>	Initialize MAC.
<code>CRYPTO_POLY1305_ARIA_Add()</code>	Add to MAC.
<code>CRYPTO_POLY1305_ARIA_Final()</code>	Compute MAC.
<code>CRYPTO_POLY1305_ARIA_Final_128()</code>	Compute MAC, fixed size.
<code>CRYPTO_POLY1305_ARIA_Kill()</code>	Destroy MAC context.
Key functions	
<code>CRYPTO_POLY1305_ARIA_Clamp()</code>	Clamp key.

6.40.1.1 CRYPTO_POLY1305_ARIA_Add()

Description

Add to MAC.

Prototype

```
void CRYPTO_POLY1305_ARIA_Add(          CRYPTO_POLY1305_ARIA_CONTEXT * pSelf,
                                     const U8                      * pInput,
                                     unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.40.1.2 CRYPTO_POLY1305_ARIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_ARIA_Calc(      U8      * pTag,
                                         unsigned   TagLen,
                                         const U8    * pKey,
                                         const U8    * pIV,
                                         const U8    * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag.
TagLen	Octet length of the requested authentication tag, at most 16 octets.
pKey	Pointer to key octet string, 32 octets.
pIV	Pointer to IV octet string, 16 octets.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.40.1.3 CRYPTO_POLY1305_ARIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_ARIA_Calc_128(      U8      * pTag,
                                         const U8      * pKey,
                                         const U8      * pIV,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.40.1.4 CRYPTO_POLY1305_ARIA_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_ARIA_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

6.40.1.5 CRYPTO_POLY1305_ARIA_Final()

Description

Compute MAC.

Prototype

```
void CRYPTO_POLY1305_ARIA_Final(CRYPTO_POLY1305_ARIA_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  unsigned OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-ARIA context.
pOutput	Pointer to object that receives the authentication tag.
OutputLen	Octet length of the requested authentication tag.

6.40.1.6 CRYPTO_POLY1305_ARIA_Final_128()

Description

Compute MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_ARIA_Final_128(CRYPTO_POLY1305_ARIA_CONTEXT * pSelf,  
                                      U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-ARIA context.
pOutput	Pointer to object that receives the authentication tag, 16 octets.

6.40.1.7 CRYPTO_POLY1305_ARIA_InitEx_256_128()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_ARIA_InitEx_256_128
(
    CRYPTO_POLY1305_ARIA_CONTEXT * pSelf,
    const U8                      * pKey,
    const U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-ARIA context.
pKey	Pointer to key octet string, 32 bytes.
pIV	Pointer to IV octet string, 16 bytes.

6.40.1.8 CRYPTO_POLY1305_ARIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_POLY1305_ARIA_Kill(CRYPTO_POLY1305_ARIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-ARIA context.

6.40.1.9 CRYPTO_POLY1305_ARIA_Verify()

Description

Verify MAC.

Prototype

```
int CRYPTO_POLY1305_ARIA_Verify(const U8      * pTag,
                                  unsigned     TagLen,
                                  const U8      * pKey,
                                  const U8      * pIV,
                                  const U8      * pInput,
                                  unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag.
TagLen	Octet length of the authentication tag.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.40.1.10 CRYPTO_POLY1305_ARIA_Verify_128()

Description

Verify MAC, fixed size.

Prototype

```
int CRYPTO_POLY1305_ARIA_Verify_128(const U8      * pTag,
                                       const U8      * pKey,
                                       const U8      * pIV,
                                       const U8      * pInput,
                                       unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.40.2 Generic API

The following table lists the Poly1305-ARIA functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_POLY1305_ARIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_POLY1305_ARIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_POLY1305_ARIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_POLY1305_ARIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_POLY1305_ARIA_Kill()</code>	Destroy MAC context.

6.40.2.1 CRYPTO_MAC_POLY1305_ARIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_POLY1305_ARIA_Add(      void      * pContext,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.40.2.2 CRYPTO_MAC_POLY1305_ARIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_POLY1305_ARIA_Final(void      * pContext,
                                       U8        * pMAC,
                                       unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.40.2.3 CRYPTO_MAC_POLY1305_ARIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_POLY1305_ARIA_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.40.2.4 CRYPTO_MAC_POLY1305_ARIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_POLY1305_ARIA_InitEx(      void      * pContext,
                                              unsigned   DigestLen,
                                              const U8   * pKey,
                                              unsigned   KeyLen,
                                              const U8   * pIV,
                                              unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.40.2.5 CRYPTO_MAC_POLY1305_ARIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_POLY1305_ARIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.41 Poly1305-Camellia

6.41.1 Type-safe API

The following table lists the Poly1305-Camellia type-safe API functions.

Function	Description
Message functions	
CRYPTO_POLY1305_CAMELLIA_Calc()	Calculate MAC.
CRYPTO_POLY1305_CAMELLIA_Calc_128()	Calculate MAC, fixed size.
CRYPTO_POLY1305_CAMELLIA_Verify()	Verify MAC.
CRYPTO_POLY1305_CAMELLIA_Verify_128()	Verify MAC, fixed size.
Incremental functions	
CRYPTO_POLY1305_CAMELLIA_InitEx_256_128()	Initialize MAC.
CRYPTO_POLY1305_CAMELLIA_Add()	Add to MAC.
CRYPTO_POLY1305_CAMELLIA_Final()	Compute MAC.
CRYPTO_POLY1305_CAMELLIA_Final_128()	Compute MAC, fixed size.
CRYPTO_POLY1305_CAMELLIA_Kill()	Destroy MAC context.
Key functions	
CRYPTO_POLY1305_CAMELLIA_Clamp()	Clamp key.

6.41.1.1 CRYPTO_POLY1305_CAMELLIA_Add()

Description

Add to MAC.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Add
(
    CRYPTO_POLY1305_CAMELLIA_CONTEXT * pSelf,
    const U8                         * pInput,
    unsigned                           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.41.1.2 CRYPTO_POLY1305_CAMELLIA_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Calc(      U8      * pTag,
                                             unsigned TagLen,
                                             const U8   * pKey,
                                             const U8   * pIV,
                                             const U8   * pInput,
                                             unsigned InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag.
TagLen	Octet length of the requested authentication tag, at most 16 octets.
pKey	Pointer to key octet string, 32 octets.
pIV	Pointer to IV octet string, 16 octets.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.41.1.3 CRYPTO_POLY1305_CAMELLIA_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Calc_128(      U8      * pTag,
                                              const U8      * pKey,
                                              const U8      * pIV,
                                              const U8      * pInput,
                                              unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.41.1.4 CRYPTO_POLY1305_CAMELLIA_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

6.41.1.5 CRYPTO_POLY1305_CAMELLIA_Final()

Description

Compute MAC.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Final(CRYPTO_POLY1305_CAMELLIA_CONTEXT * pSelf,  
                                     U8  
                                     unsigned  
                                     * pOutput,  
                                     OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Camellia context.
pOutput	Pointer to object that receives the authentication tag.
OutputLen	Octet length of the requested authentication tag.

6.41.1.6 CRYPTO_POLY1305_CAMELLIA_Final_128()

Description

Compute MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Final_128
    (CRYPTO_POLY1305_CAMELLIA_CONTEXT * pSelf,
     U8                                * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Camellia context.
pOutput	Pointer to object that receives the authentication tag, 16 octets.

6.41.1.7 CRYPTO_POLY1305_CAMELLIA_InitEx_256_128()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_InitEx_256_128
(
    CRYPTO_POLY1305_CAMELLIA_CONTEXT * pSelf,
    const U8                         * pKey,
    const U8                         * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Camellia context.
pKey	Pointer to key octet string, 32 bytes.
pIV	Pointer to IV octet string, 16 bytes.

6.41.1.8 CRYPTO_POLY1305_CAMELLIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_POLY1305_CAMELLIA_Kill(CRYPTO_POLY1305_CAMELLIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Camellia context.

6.41.1.9 CRYPTO_POLY1305_CAMELLIA_Verify()

Description

Verify MAC.

Prototype

```
int CRYPTO_POLY1305_CAMELLIA_Verify(const U8      * pTag,
                                      unsigned     TagLen,
                                      const U8      * pKey,
                                      const U8      * pIV,
                                      const U8      * pInput,
                                      unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag.
TagLen	Octet length of the authentication tag.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.41.1.10 CRYPTO_POLY1305_CAMELLIA_Verify_128()

Description

Verify MAC, fixed size.

Prototype

```
int CRYPTO_POLY1305_CAMELLIA_Verify_128(const U8 * pTag,
                                         const U8 * pKey,
                                         const U8 * pIV,
                                         const U8 * pInput,
                                         unsigned InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.41.2 Generic API

The following table lists the Poly1305-Camellia functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_POLY1305_CAMELLIA_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_POLY1305_CAMELLIA_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_POLY1305_CAMELLIA_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_POLY1305_CAMELLIA_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_POLY1305_CAMELLIA_Kill()</code>	Destroy MAC context.

6.41.2.1 CRYPTO_MAC_POLY1305_CAMELLIA_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_POLY1305_CAMELLIA_Add(      void      * pContext,
                                              const U8      * pInput,
                                              unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.41.2.2 CRYPTO_MAC_POLY1305_CAMELLIA_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_POLY1305_CAMELLIA_Final(void      * pContext,
                                         U8        * pMAC,
                                         unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.41.2.3 CRYPTO_MAC_POLY1305_CAMELLIA_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_POLY1305_CAMELLIA_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.41.2.4 CRYPTO_MAC_POLY1305_CAMELLIA_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_POLY1305_CAMELLIA_InitEx(      void      * pContext,
                                                unsigned   DigestLen,
                                                const U8    * pKey,
                                                unsigned   KeyLen,
                                                const U8    * pIV,
                                                unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.41.2.5 CRYPTO_MAC_POLY1305_CAMELLIA_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_POLY1305_CAMELLIA_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.42 Poly1305-Twofish

6.42.1 Type-safe API

The following table lists the Poly1305-Twofish type-safe API functions.

Function	Description
Message functions	
CRYPTO_POLY1305_TWOFISH_Calc()	Calculate MAC.
CRYPTO_POLY1305_TWOFISH_Calc_128()	Calculate MAC, fixed size.
CRYPTO_POLY1305_TWOFISH_Verify()	Verify MAC.
CRYPTO_POLY1305_TWOFISH_Verify_128()	Verify MAC, fixed size.
Incremental functions	
CRYPTO_POLY1305_TWOFISH_InitEx_256_128()	Initialize MAC.
CRYPTO_POLY1305_TWOFISH_Add()	Add to MAC.
CRYPTO_POLY1305_TWOFISH_Final()	Compute MAC.
CRYPTO_POLY1305_TWOFISH_Final_128()	Compute MAC, fixed size.
CRYPTO_POLY1305_TWOFISH_Kill()	Destroy MAC context.
Key functions	
CRYPTO_POLY1305_TWOFISH_Clamp()	Clamp key.

6.42.1.1 CRYPTO_POLY1305_TWOFISH_Add()

Description

Add to MAC.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Add(          CRYPTO_POLY1305_TWOFISH_CONTEXT * pSelf,
                                         const U8                      * pInput,
                                         unsigned                     InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.42.1.2 CRYPTO_POLY1305_TWOFISH_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Calc(      U8      * pTag,
                                         unsigned TagLen,
                                         const U8      * pKey,
                                         const U8      * pIV,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag.
TagLen	Octet length of the requested authentication tag, at most 16 octets.
pKey	Pointer to key octet string, 32 octets.
pIV	Pointer to IV octet string, 16 octets.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.42.1.3 CRYPTO_POLY1305_TWOFISH_Calc_128()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Calc_128(      U8      * pTag,
                                              const U8      * pKey,
                                              const U8      * pIV,
                                              const U8      * pInput,
                                              unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that receives the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

6.42.1.4 CRYPTO_POLY1305_TWOFISH_Clamp()

Description

Clamp key.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Clamp(U8 * pKey);
```

Parameters

Parameter	Description
pKey	Pointer to key to clamp, 32 octets.

6.42.1.5 CRYPTO_POLY1305_TWOFISH_Final()

Description

Compute MAC.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Final(CRYPTO_POLY1305_TWOFISH_CONTEXT * pSelf,  
                                     U8  
                                     unsigned  
                                     pOutput,  
                                     OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Twofish context.
pOutput	Pointer to object that receives the authentication tag.
OutputLen	Octet length of the requested authentication tag.

6.42.1.6 CRYPTO_POLY1305_TWOFISH_Final_128()

Description

Compute MAC, fixed size.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Final_128(CRYPTO_POLY1305_TWOFISH_CONTEXT * pSelf,  
                                         U8 * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Twofish context.
pOutput	Pointer to object that receives the authentication tag, 16 octets.

6.42.1.7 CRYPTO_POLY1305_TWOFISH_InitEx_256_128()

Description

Initialize MAC.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_InitEx_256_128
    (
        CRYPTO_POLY1305_TWOFISH_CONTEXT * pSelf,
        const U8                         * pKey,
        const U8                         * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Poly1305-Twofish context.
pKey	Pointer to key octet string, 32 bytes.
pIV	Pointer to IV octet string, 16 bytes.

6.42.1.8 CRYPTO_POLY1305_TWOFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_POLY1305_TWOFISH_Kill(CRYPTO_POLY1305_TWOFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to Poly1305-Twofish context.

6.42.1.9 CRYPTO_POLY1305_TWOFISH_Verify()

Description

Verify MAC.

Prototype

```
int CRYPTO_POLY1305_TWOFISH_Verify(const U8      * pTag,
                                     unsigned   TagLen,
                                     const U8      * pKey,
                                     const U8      * pIV,
                                     const U8      * pInput,
                                     unsigned   InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag.
TagLen	Octet length of the authentication tag.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.42.1.10 CRYPTO_POLY1305_TWOFISH_Verify_128()

Description

Verify MAC, fixed size.

Prototype

```
int CRYPTO_POLY1305_TWOFISH_Verify_128(const U8      * pTag,
                                         const U8      * pKey,
                                         const U8      * pIV,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pTag	Pointer to object that contains the authentication tag, 16 octets.
pKey	Pointer to key octet string.
pIV	Pointer to IV octet string.
pInput	Pointer to the message octet string.
InputLen	Octet length of the message octet string.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

6.42.2 Generic API

The following table lists the Poly1305-Twofish functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_POLY1305_TWOFISH_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_POLY1305_TWOFISH_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_POLY1305_TWOFISH_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_POLY1305_TWOFISH_Final_128()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_POLY1305_TWOFISH_Kill()</code>	Destroy MAC context.

6.42.2.1 CRYPTO_MAC_POLY1305_TWOFISH_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_POLY1305_TWOFISH_Add(      void      * pContext,
                                             const U8      * pInput,
                                             unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.42.2.2 CRYPTO_MAC_POLY1305_TWOFISH_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_POLY1305_TWOFISH_Final(void * pContext,  
                                         U8 * pMAC,  
                                         unsigned MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.42.2.3 CRYPTO_MAC_POLY1305_TWOFISH_Final_128()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_POLY1305_TWOFISH_Final_128(void * pContext,  
                                         U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.42.2.4 CRYPTO_MAC_POLY1305_TWOFISH_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_POLY1305_TWOFISH_InitEx(      void      * pContext,
                                                unsigned   DigestLen,
                                                const U8   * pKey,
                                                unsigned   KeyLen,
                                                const U8   * pIV,
                                                unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.42.2.5 CRYPTO_MAC_POLY1305_TWOFISH_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_POLY1305_TWOFISH_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.43 Michael

6.43.1 Type-safe API

The following table lists the Michael type-safe API functions.

Function	Description
Message functions	
<code>CRYPTO_MICHAEL_Calc()</code>	Calculate MAC.
<code>CRYPTO_MICHAEL_Calc_64()</code>	Calculate MAC, fixed size.
Incremental functions	
<code>CRYPTO_MICHAEL_Init()</code>	Initialize context.
<code>CRYPTO_MICHAEL_Init_64()</code>	Initialize context, fixed size.
<code>CRYPTO_MICHAEL_Add()</code>	Add data to MAC.
<code>CRYPTO_MICHAEL_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MICHAEL_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MICHAEL_Kill()</code>	Destroy context.

6.43.1.1 CRYPTO_MICHAEL_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MICHAEL_Add(      CRYPTO_MICHAEL_CONTEXT * pSelf,
                               const U8           * pInput,
                               unsigned          InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pInput	Pointer to octet string to add to MAC.
InputLen	Octet length of the octet string.

Additional information

The input data can be any length and is not limited to the underlying block size: the algorithm internally manages correct blocking of data.

6.43.1.2 CRYPTO_MICHAEL_Calc()

Description

Calculate MAC.

Prototype

```
void CRYPTO_MICHAEL_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8   * pKey,
                                unsigned   KeyLen,
                                const U8   * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.43.1.3 CRYPTO_MICHAEL_Calc_64()

Description

Calculate MAC, fixed size.

Prototype

```
void CRYPTO_MICHAEL_Calc_64(      U8      * pOutput,
                                    const U8      * pKey,
                                    unsigned      KeyLen,
                                    const U8      * pInput,
                                    unsigned      InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the MAC, 8 octets.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

6.43.1.4 CRYPTO_MICHAEL_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MICHAEL_Final(CRYPTO_MICHAEL_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC.
OutputLen	Octet length of the MAC.

Additional information

It is possible to truncate the MAC by specifying `OutputLen` less than the full digest length: in this case, the leftmost (most significant) octets of the MAC are written to the receiving object.

6.43.1.5 CRYPTO_MICHAEL_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MICHAEL_Final_64(CRYPTO_MICHAEL_CONTEXT * pSelf,  
                           U8                      * pOutput);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pOutput	Pointer to object that receives the MAC, 8 octets.

6.43.1.6 CRYPTO_MICHAEL_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MICHAEL_Init(      CRYPTO_MICHAEL_CONTEXT * pSelf,
                               const U8                  * pKey,
                               unsigned                 KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key.
KeyLen	Octet length of the cipher key.

6.43.1.7 CRYPTO_MICHAEL_Init_64()

Description

Initialize context, fixed size.

Prototype

```
void CRYPTO_MICHAEL_Init_64(          CRYPTO_MICHAEL_CONTEXT * pSelf,  
                                const U8                      * pKey);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.
pKey	Pointer to cipher key, 8 octets.

6.43.1.8 CRYPTO_MICHAEL_Kill()

Description

Destroy context.

Prototype

```
void CRYPTO_MICHAEL_Kill(CRYPTO_MICHAEL_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MAC context.

6.43.2 Generic API

The following table lists the Michael functions that conform to the generic MAC API.

Function	Description
<code>CRYPTO_MAC_MICHAEL_Init()</code>	Initialize context.
<code>CRYPTO_MAC_MICHAEL_InitEx()</code>	Initialize context, include subkey.
<code>CRYPTO_MAC_MICHAEL_Add()</code>	Add data to MAC.
<code>CRYPTO_MAC_MICHAEL_Final()</code>	Finish MAC calculation.
<code>CRYPTO_MAC_MICHAEL_Final_64()</code>	Finish MAC calculation, fixed size.
<code>CRYPTO_MAC_MICHAEL_Kill()</code>	Destroy MAC context.

6.43.2.1 CRYPTO_MAC_MICHAEL_Add()

Description

Add data to MAC.

Prototype

```
void CRYPTO_MAC_MICHAEL_Add(      void      * pContext,
                                    const U8      * pInput,
                                    unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pInput	Pointer to input to add to MAC.
InputLen	Octet length of the input string.

6.43.2.2 CRYPTO_MAC_MICHAEL_Final()

Description

Finish MAC calculation.

Prototype

```
void CRYPTO_MAC_MICHAEL_Final(void      * pContext,
                               U8        * pMAC,
                               unsigned   MACLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.
MACLen	Octet length of the MAC.

6.43.2.3 CRYPTO_MAC_MICHAEL_Final_64()

Description

Finish MAC calculation, fixed size.

Prototype

```
void CRYPTO_MAC_MICHAEL_Final_64(void * pContext,  
                                  U8      * pMAC);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pMAC	Pointer to object that receives the MAC.

6.43.2.4 CRYPTO_MAC_MICHAEL_Init()

Description

Initialize context.

Prototype

```
void CRYPTO_MAC_MICHAEL_Init(      void      * pContext,
                                     const U8      * pKey,
                                     unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
pKey	Pointer to octet string that is the key.
KeyLen	Length of key octet string.

6.43.2.5 CRYPTO_MAC_MICHAEL_InitEx()

Description

Initialize context, include subkey.

Prototype

```
void CRYPTO_MAC_MICHAEL_InitEx(      void      * pContext,
                                         unsigned   DigestLen,
                                         const U8    * pKey,
                                         unsigned   KeyLen,
                                         const U8    * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.
DigestLen	Octet length of the digest octet string.
pKey	Pointer to key octet string.
KeyLen	Octet length of the key octet string.
pIV	Pointer to IV octet string.
IVLen	Octet length of the IV octet string.

6.43.2.6 CRYPTO_MAC_MICHAEL_Kill()

Description

Destroy MAC context.

Prototype

```
void CRYPTO_MAC_MICHAEL_Kill(void * pContext);
```

Parameters

Parameter	Description
pContext	Pointer to MAC context.

6.43.3 Self-test API

The following table lists the MICHAEL self-test API functions.

Function	Description
<code>CRYPTO_MICHAEL_802v11_SelfTest()</code>	Run Michael test vectors from 802.11-2016.

6.43.3.1 CRYPTO_MICHAEL_802v11_SelfTest()

Description

Run Michael test vectors from 802.11-2016.

Prototype

```
void CRYPTO_MICHAEL_802v11_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

Chapter 7

Symmetric encryption (secret key)

emCrypt implements the following ciphers:

- DES
- AES
- IDEA
- SEED
- ARIA
- Camellia
- CAST
- ChaCha20
- Blowfish
- Twofish
- PRESENT
- RC4

7.1 Introduction

In general a symmetric encryption or decryption is performed in two steps:

- Initializing using the cipher key. This will determine the direction of the operation (encryption or decryption).
- Encrypting or decrypting data.

The initialization prepares the key for the operation and stores it into a data structure called a *cipher context*. The cipher context is maintained by the cipher functions, only the memory must be provided by the caller. It can be used for multiple encryption or decryption operations with the same key and may be discarded if the key is no longer used. Encryption and decryption can not be intermixed with the same cipher context.

The API functions are named in the same way for all cipher algorithms:

- CRYPTO_<cipher_name_and_mode>_InitEncrypt() for initializing and preparing the key for encryption operations.
- CRYPTO_<cipher_name_and_mode>_Encrypt() to encrypt data.

Respectively:

- CRYPTO_<cipher_name_and_mode>_InitDecrypt() for initializing and preparing the key for decryption operations.
- CRYPTO_<cipher_name_and_mode>_Decrypt() to decrypt data.

Example

```
//  
// Example for an AES encryption.  
//  
static const U8          Key[16] = { 0x08, 0x15, 0x85, 0x11, ..., 0x5b, 0xa3 };  
CRYPTO_AES_CONTEXT        AES_Context;  
//  
// Prepare the key for encryption.  
//  
CRYPTO_AES_InitEncrypt(&AES_Context, Key, sizeof(Key));  
//  
// Encrypt data.  
//  
CRYPTO_AES_ECB_Encrypt(&AES_Context, pChiperData, pClearData, DataLen);  
//  
// Encrypt more data.  
//  
CRYPTO_AES_ECB_Encrypt(&AES_Context, pChiperData2, pClearData2, Data2Len);  
//  
// From now, AES_Context is not used any more.  
// For security reasons, clear the key from memory.  
//  
CRYPTO_AES_Kill(&AES_Context);
```

Besides the type-safe API functions described above, there are also generic API functions, that use a void pointer to take the cipher context. These are useful, if the API functions shall be called via functions pointers to dynamically choose different cipher algorithms. When using the generic functions the caller is responsible to provide the correct context (or memory areas) via the void pointer argument.

7.2 DES

7.2.1 Standards reference

DES is specified by the following document:

- FIPS PUB 46-3 — *Data Encryption Standard (DES)*

7.2.2 Algorithm parameters

7.2.2.1 Block size

```
#define CRYPTO_TDES_BLOCK_BYTE_COUNT 16
```

The number of bytes in a single TDES block.

7.2.2.2 Key size

```
#define CRYPTO_TDES_1KEY_SIZE 8  
#define CRYPTO_TDES_2KEY_SIZE 16  
#define CRYPTO_TDES_3KEY_SIZE 24
```

The number of bytes for three TDES keying options:

Keying mode	Description
CRYPTO_TDES_1KEY_SIZE	All three keys are identical with K1 = K2 = K3.
CRYPTO_TDES_2KEY_SIZE	K1 and K2 are independent and K3 = K1.
CRYPTO_TDES_3KEY_SIZE	All three keys are independent.

7.2.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_DES_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize DES and 3DES to place tables in RAM rather than flash. Optimization levels are 0 through 5 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.38 KB	Flash	2.1 KB	1.3 KB	3.4 KB
1	0.38 KB	Flash	2.1 KB	2.1 KB	4.2 KB
2	0.38 KB	Flash	2.1 KB	5.3 KB	7.4 KB
3	0.38 KB	RAM	2.1 KB	1.3 KB	3.4 KB
4	0.38 KB	RAM	2.1 KB	2.1 KB	4.2 KB
5	0.38 KB	RAM	2.1 KB	5.3 KB	7.4 KB

7.2.4 Type-safe API

The following table lists the DES type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_TDES_Install()</code>	Install cipher.
<code>CRYPTO_TDES_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_TDES_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_TDES_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_TDES_InitEncryptEx()</code>	Initialize, expand key, encrypt mode.
<code>CRYPTO_TDES_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_TDES_InitDecryptEx()</code>	Initialize, expand key, decrypt mode.
<code>CRYPTO_TDES_Kill()</code>	Clear TDES context.
Single blocks	
<code>CRYPTO_TDES_Encrypt()</code>	Encrypt block.
<code>CRYPTO_TDES_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_TDES_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_TDES_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_TDES_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_TDES_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_TDES_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_TDES_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_TDES_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_TDES_CTR_Decrypt()</code>	Decrypt, CTR mode.
Formatting	
<code>CRYPTO_TDES_CheckParity()</code>	Check parity of DES key.
<code>CRYPTO_TDES_CorrectParity()</code>	Correct parity of DES key.
<code>CRYPTO_TDES_InsertParity()</code>	Insert parity bits into key.

7.2.4.1 CRYPTO_TDES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_TDES_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.2.4.2 CRYPTO_TDES_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_TDES_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.2.4.3 CRYPTO_TDES_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_TDES_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                               const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.2.4.4 CRYPTO_TDES_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_TDES_InitEncrypt(      CRYPTO_TDES_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.4.5 CRYPTO_TDES_InitEncryptEx()

Description

Initialize, expand key, encrypt mode.

Prototype

```
void CRYPTO_TDES_InitEncryptEx(      CRYPTO_TDES_CONTEXT * pSelf,
                                      const U8           * pKey,
                                      unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.4.6 CRYPTO_TDES_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_TDES_InitDecrypt(      CRYPTO_TDES_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.4.7 CRYPTO_TDES_InitDecryptEx()

Description

Initialize, expand key, decrypt mode.

Prototype

```
void CRYPTO_TDES_InitDecryptEx(      CRYPTO_TDES_CONTEXT * pSelf,
                                      const U8           * pKey,
                                      unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.4.8 CRYPTO_TDES_Kill()

Description

Clear TDES context.

Prototype

```
void CRYPTO_TDES_Kill(CRYPTO_TDES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.2.4.9 CRYPTO_TDES_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_TDES_Encrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                               U8           * pOutput,
                               const U8      * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.2.4.10 CRYPTO_TDES_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_TDES_Decrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                               U8           * pOutput,
                           const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.2.4.11 CRYPTO_TDES_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_TDES_ECB_Encrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.2.4.12 CRYPTO_TDES_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_TDES_ECB_Decrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.2.4.13 CRYPTO_TDES_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_TDES_CBC_Encrypt( CRYPTO_TDES_CONTEXT * pSelf,  
                               U8 * pOutput,  
                               const U8 * pInput,  
                               unsigned InputLen,  
                               U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.2.4.14 CRYPTO_TDES_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_TDES_CBC_Decrypt( CRYPTO_TDES_CONTEXT * pSelf,  
                               U8 * pOutput,  
                               const U8 * pInput,  
                               unsigned InputLen,  
                               U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.2.4.15 CRYPTO_TDES_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_TDES_OFB_Encrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.2.4.16 CRYPTO_TDES_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_TDES_OFB_Decrypt(          CRYPTO_TDES_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.2.4.17 CRYPTO_TDES_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_TDES_CTR_Encrypt( CRYPTO_TDES_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.2.4.18 CRYPTO_TDES_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_TDES_CTR_Decrypt( CRYPTO_TDES_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized TDES context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.2.4.19 CRYPTO_TDES_CheckParity()

Description

Check parity of DES key.

Prototype

```
int CRYPTO_TDES_CheckParity(const U8      * pKey,  
                           unsigned KeyLen);
```

Parameters

Parameter	Description
pKey	Pointer to key.
KeyLen	Octet length of the key.

Return value

≥ 0 Success, parity is correct.
< 0 Failure, at least one parity bit in error.

Additional information

The low-order bit of each key byte contains the parity.

7.2.4.20 CRYPTO_TDES_CorrectParity()

Description

Correct parity of DES key.

Prototype

```
void CRYPTO_TDES_CorrectParity(U8      * pKey,
                               unsigned KeyLen);
```

Parameters

Parameter	Description
pKey	Pointer to key.
KeyLen	Octet length of the key.

Additional information

The low-order bits of each key byte are corrected to odd parity.

7.2.4.21 CRYPTO_TDES_InsertParity()

Description

Insert parity bits into key.

Prototype

```
unsigned CRYPTO_TDES_InsertParity(      U8      * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to expanded key with odd parity inserted.
pInput	Pointer to input key without parity.
InputLen	Octet length of the input key.

Return value

Octet length of the expanded key.

Additional information

The input key, which has no parity bits, is expanded to a longer key with the low-order bits of each expanded octet set to odd parity.

The number of output bytes is $8 * (\text{InputLen}/7)$ so a 21-octet TDES key will expand to a 24-octet TDES key with parity inserted.

7.2.5 Generic API

The following table lists the TDES functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_TDES_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_TDES_64_InitEncrypt()</code>	Initialize, encrypt mode, 64-bit key.
<code>CRYPTO_CIPHER_TDES_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_TDES_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_TDES_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_TDES_64_InitDecrypt()</code>	Initialize, decrypt mode, 64-bit key.
<code>CRYPTO_CIPHER_TDES_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_TDES_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
Single blocks	
<code>CRYPTO_CIPHER_TDES_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CIPHER_TDES_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CIPHER_TDES_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_TDES_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_TDES_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_TDES_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_TDES_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_TDES_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_TDES_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_TDES_CTR_0_8_Encrypt()</code>	Encrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_TDES_CTR_4_4_Encrypt()</code>	Encrypt, CTR(4,4) mode.
<code>CRYPTO_CIPHER_TDES_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_TDES_CTR_0_8_Decrypt()</code>	Decrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_TDES_CTR_4_4_Decrypt()</code>	Decrypt, CTR(4,4) mode.

7.2.5.1 CRYPTO_CIPHER_TDES_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_TDES_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.5.2 CRYPTO_CIPHER_TDES_64_InitEncrypt()

Description

Initialize, encrypt mode, 64-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_64_InitEncrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.3 CRYPTO_CIPHER_TDES_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_128_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.4 CRYPTO_CIPHER_TDES_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_192_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.5 CRYPTO_CIPHER_TDES_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_TDES_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.2.5.6 CRYPTO_CIPHER_TDES_64_InitDecrypt()

Description

Initialize, decrypt mode, 64-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_64_InitDecrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.7 CRYPTO_CIPHER_TDES_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_128_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.8 CRYPTO_CIPHER_TDES_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_TDES_192_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pKey	Pointer to key.

7.2.5.9 CRYPTO_CIPHER_TDES_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_TDES_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.2.5.10 CRYPTO_CIPHER_TDES_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_TDES_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.2.5.11 CRYPTO_CIPHER_TDES_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_TDES_ECB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.2.5.12 CRYPTO_CIPHER_TDES_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_TDES_ECB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.2.5.13 CRYPTO_CIPHER_TDES_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CBC_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.2.5.14 CRYPTO_CIPHER_TDES_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CBC_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.2.5.15 CRYPTO_CIPHER_TDES_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_TDES_OFB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.2.5.16 CRYPTO_CIPHER_TDES_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_TDES_OFB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.2.5.17 CRYPTO_CIPHER_TDES_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_Encrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.2.5.18 CRYPTO_CIPHER_TDES_CTR_0_8_Encrypt()

Description

Encrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_0_8_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.2.5.19 CRYPTO_CIPHER_TDES_CTR_4_4_Encrypt()

Description

Encrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_4_4_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[4...7].

7.2.5.20 CRYPTO_CIPHER_TDES_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_Decrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.2.5.21 CRYPTO_CIPHER_TDES_CTR_0_8_Decrypt()

Description

Decrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_0_8_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.2.5.22 CRYPTO_CIPHER_TDES_CTR_4_4_Decrypt()

Description

Decrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_TDES_CTR_4_4_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TDES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[4...7].

7.2.6 Self-test API

The following table lists the TDES self-test API functions.

Function	Description
<code>CRYPTO_TDES_ECB_CAVS_SelfTest()</code>	Run CAVS TDES self-test.
<code>CRYPTO_TDES_CBC_CAVS_SelfTest()</code>	Run CAVS TDES self-test.

7.2.6.1 CRYPTO_TDES_ECB_CAVS_SelfTest()

Description

Run CAVS TDES self-test.

Prototype

```
void CRYPTO_TDES_ECB_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.2.6.2 CRYPTO_TDES_CBC_CAVS_SelfTest()

Description

Run CAVS TDES self-test.

Prototype

```
void CRYPTO_TDES_CBC_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3 AES

7.3.1 Standards reference

AES is specified by the following document:

- FIPS PUB 197 — *Specification for the Advanced Encryption Standard (AES)*

7.3.2 Algorithm parameters

7.3.2.1 Block size

```
#define CRYPTO_AES_BLOCK_SIZE    16
```

The number of bytes in a single AES block.

7.3.2.2 Key size

```
#define CRYPTO_AES128_KEY_SIZE   16  
#define CRYPTO_AES192_KEY_SIZE   24  
#define CRYPTO_AES256_KEY_SIZE   32
```

The number of bytes for each of the supported key sizes.

7.3.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_AES_OPTIMIZE 2
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize AES to use tables for matrix multiplication. Optimization levels are 0 through 7 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.24 KB	Flash	2.0 KB	3.2 KB	5.2 KB
1	0.24 KB	Flash	2.0 KB	2.7 KB	4.7 KB
2	0.24 KB	Flash	8.5 KB	2.4 KB	10.9 KB
3	0.24 KB	Flash	1.9 KB	12.5 KB	14.4 KB
4	0.24 KB	RAM	2.0 KB	3.2 KB	5.2 KB
5	0.24 KB	RAM	2.0 KB	2.7 KB	4.7 KB
6	0.24 KB	RAM	8.5 KB	2.4 KB	10.9 KB
7	0.24 KB	RAM	1.9 KB	12.5 KB	14.4 KB

7.3.4 Hardware acceleration

The following processors provide hardware acceleration for AES:

- NXP LPC18Sxx and LPC43Sxx, see *LPC18S and LPC43S AES ROM (Add-on)* on page 2654.
- NXP Kinetis, see *Kinetis CAU coprocessor (Add-on)* on page 2651.
- STMicroelectronics STM32F4 and STM32F7, see *STM32 CRYP coprocessor (Add-on)* on page 2660.
- STMicroelectronics STM32L4 see *STM32 AES coprocessor (Add-on)* on page 2659.
- EFM32 Pearl and Jade Gecko, see *EFM32 CRYPTO coprocessor (Add-on)* on page 2637.

7.3.5 Type-safe API

The following table lists the AES type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_AES_Install()</code>	Install cipher.
<code>CRYPTO_AES_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_AES_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_AES_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_AES_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_AES_Kill()</code>	Clear AES context.
Single-block AES	
<code>CRYPTO_AES_Encrypt()</code>	Encrypt block.
<code>CRYPTO_AES_Decrypt()</code>	Decrypt block.
Basic cipher modes	
<code>CRYPTO_AES_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_AES_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_AES_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_AES_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_AES_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_AES_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_AES_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_AES_CTR_Decrypt()</code>	Decrypt, CTR mode.
AEAD modes	
<code>CRYPTO_AES_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_AES_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_AES_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_AES_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.3.5.1 CRYPTO_AES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_AES_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                        const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.3.5.2 CRYPTO_AES_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_AES_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.3.5.3 CRYPTO_AES_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_AES_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                           const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.3.5.4 CRYPTO_AES_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_AES_InitEncrypt(      CRYPTO_AES_CONTEXT * pSelf,
                                  const U8          * pKey,
                                  unsigned         KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.3.5.5 CRYPTO_AES_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_AES_InitDecrypt(      CRYPTO_AES_CONTEXT * pSelf,
                                  const U8          * pKey,
                                  unsigned         KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.3.5.6 CRYPTO_AES_Kill()

Description

Clear AES context.

Prototype

```
void CRYPTO_AES_Kill(CRYPTO_AES_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.3.5.7 CRYPTO_AES_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_AES_Encrypt(      CRYPTO_AES_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.3.5.8 CRYPTO_AES_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_AES_Decrypt(      CRYPTO_AES_CONTEXT * pSelf,
                               U8                  * pOutput,
                           const U8              * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.3.5.9 CRYPTO_AES_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_AES_ECB_Encrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.3.5.10 CRYPTO_AES_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_AES_ECB_Decrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                  U8           * pOutput,
const U8           * pInput,
      unsigned        InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.3.5.11 CRYPTO_AES_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_AES_CBC_Encrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                    U8           * pOutput,
        const U8           * pInput,
        unsigned           InputLen,
        U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.3.5.12 CRYPTO_AES_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_AES_CBC_Decrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                  U8           * pOutput,
const U8           * pInput,
unsigned          InputLen,
          U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.3.5.13 CRYPTO_AES_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_AES_OFB_Encrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen,
                                    U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.3.5.14 CRYPTO_AES_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_AES_OFB_Decrypt(          CRYPTO_AES_CONTEXT * pSelf,
                                  U8           * pOutput,
const U8           * pInput,
unsigned          InputLen,
          U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.3.5.15 CRYPTO_AES_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_AES_CTR_Encrypt( CRYPTO_AES_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pInput,
                             unsigned InputLen,
                             U8 * pCTR,
                             unsigned CTRIndex,
                             unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.3.5.16 CRYPTO_AES_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_AES_CTR_Decrypt( CRYPTO_AES_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pInput,
                             unsigned InputLen,
                             U8 * pCTR,
                             unsigned CTRIndex,
                             unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized AES context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.3.5.17 CRYPTO_AES_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_AES_CCM_Encrypt( CRYPTO_AES_CONTEXT * pSelf,
                             U8          * pOutput,
                             U8          * pTag,
                             TagLen,
                             const U8      * pInput,
                             InputLen,
                             const U8      * pAAD,
                             AADLen,
                             const U8      * pIV,
                             IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

7.3.5.18 CRYPTO_AES_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_AES_CCM_Decrypt( CRYPTO_AES_CONTEXT * pSelf,
                            U8 * pOutput,
                            const U8 * pTag,
                            TagLen,
                            const U8 * pInput,
                            InputLen,
                            const U8 * pAAD,
                            AADLen,
                            const U8 * pIV,
                            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- # 0 Calculated tag and given tag are not identical.

7.3.5.19 CRYPTO_AES_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_AES_GCM_Encrypt( CRYPTO_AES_CONTEXT * pSelf,
                             U8          * pOutput,
                             U8          * pTag,
                             unsigned     TagLen,
                             const U8    * pInput,
                             unsigned     InputLen,
                             const U8    * pAAD,
                             unsigned     AADLen,
                             const U8    * pIV,
                             unsigned     IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the initialization vector.

7.3.5.20 CRYPTO_AES_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_AES_GCM_Decrypt( CRYPTO_AES_CONTEXT * pSelf,
                            U8 * pOutput,
                            const U8 * pTag,
                            TagLen,
                            const U8 * pInput,
                            InputLen,
                            const U8 * pAAD,
                            AADLen,
                            const U8 * pIV,
                            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.3.6 Generic API

The following table lists the AES functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_AES_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_AES_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_AES_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_AES_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_AES_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_AES_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_AES_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_AES_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Basic cipher modes	
<code>CRYPTO_CIPHER_AES_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_AES_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_AES_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_AES_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_AES_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_AES_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_AES_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_AES_CTR_0_16_Encrypt()</code>	Encrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_AES_CTR_12_4_Encrypt()</code>	Encrypt, CTR(12,4) mode.
<code>CRYPTO_CIPHER_AES_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_AES_CTR_0_16_Decrypt()</code>	Decrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_AES_CTR_12_4_Decrypt()</code>	Decrypt, CTR(12,4) mode.
AEAD modes	
<code>CRYPTO_CIPHER_AES_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_CIPHER_AES_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_CIPHER_AES_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_CIPHER_AES_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.3.6.1 CRYPTO_CIPHER_AES_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_AES_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.3.6.2 CRYPTO_CIPHER_AES_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_128_InitEncrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.3 CRYPTO_CIPHER_AES_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_192_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.4 CRYPTO_CIPHER_AES_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_256_InitEncrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.5 CRYPTO_CIPHER_AES_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_AES_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.3.6.6 CRYPTO_CIPHER_AES_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_128_InitDecrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.7 CRYPTO_CIPHER_AES_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_192_InitDecrypt(      void * pContext,
                                             const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.8 CRYPTO_CIPHER_AES_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_AES_256_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pKey	Pointer to key.

7.3.6.9 CRYPTO_CIPHER_AES_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_AES_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.3.6.10 CRYPTO_CIPHER_AES_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_AES_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8  * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to AES context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.3.6.11 CRYPTO_CIPHER_AES_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_AES_ECB_Encrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.3.6.12 CRYPTO_CIPHER_AES_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_AES_ECB_Decrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.3.6.13 CRYPTO_CIPHER_AES_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_AES_CBC_Encrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned   InputLen,
                                         U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.3.6.14 CRYPTO_CIPHER_AES_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_AES_CBC_Decrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned   InputLen,
                                         U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.3.6.15 CRYPTO_CIPHER_AES_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_Encrypt( void      * pContext,
                                     U8        * pOutput,
                                     const U8   * pInput,
                                     unsigned   InputLen,
                                     U8        * pCTR,
                                     unsigned   CTRIndex,
                                     unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.3.6.16 CRYPTO_CIPHER_AES_CTR_0_16_Encrypt()

Description

Encrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_0_16_Encrypt(  
    void      * pContext,  
    U8       * pOutput,  
    const U8   * pInput,  
    unsigned InputLen,  
    U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[0...15].

7.3.6.17 CRYPTO_CIPHER_AES_CTR_12_4_Encrypt()

Description

Encrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_12_4_Encrypt(  
    void      * pContext,  
    U8       * pOutput,  
    const U8   * pInput,  
    unsigned InputLen,  
    U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[12...15]`.

7.3.6.18 CRYPTO_CIPHER_AES_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_Decrypt( void      * pContext,
                                     U8        * pOutput,
                                     const U8   * pInput,
                                     unsigned   InputLen,
                                     U8        * pCTR,
                                     unsigned   CTRIndex,
                                     unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.3.6.19 CRYPTO_CIPHER_AES_CTR_0_16_Decrypt()

Description

Decrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_0_16_Decrypt(  
    void      * pContext,  
    U8       * pOutput,  
    const U8   * pInput,  
    unsigned InputLen,  
    U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[0...15].

7.3.6.20 CRYPTO_CIPHER_AES_CTR_12_4_Decrypt()

Description

Decrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_AES_CTR_12_4_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.3.6.21 CRYPTO_CIPHER_AES_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CIPHER_AES_CCM_Encrypt( void      * pContext,
                                     U8        * pOutput,
                                     U8        * pTag,
                                     unsigned   TagLen,
                                     const U8   * pInput,
                                     unsigned   InputLen,
                                     const U8   * pAAD,
                                     unsigned   AADLen,
                                     const U8   * pIV,
                                     unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.3.6.22 CRYPTO_CIPHER_AES_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CIPHER_AES_CCM_Decrypt( void      * pContext,
                                    U8        * pOutput,
                                    const U8   * pTag,
                                    unsigned   TagLen,
                                    const U8   * pInput,
                                    unsigned   InputLen,
                                    const U8   * pAAD,
                                    unsigned   AADLen,
                                    const U8   * pIV,
                                    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.3.6.23 CRYPTO_CIPHER_AES_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CIPHER_AES_GCM_Encrypt( void      * pContext,
                                     U8        * pOutput,
                                     U8        * pTag,
                                     unsigned   TagLen,
                                     const U8   * pInput,
                                     unsigned   InputLen,
                                     const U8   * pAAD,
                                     unsigned   AADLen,
                                     const U8   * pIV,
                                     unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

7.3.6.24 CRYPTO_CIPHER_AES_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CIPHER_AES_GCM_Decrypt( void      * pContext,
                                    U8        * pOutput,
                                    const U8   * pTag,
                                    unsigned   TagLen,
                                    const U8   * pInput,
                                    unsigned   InputLen,
                                    const U8   * pAAD,
                                    unsigned   AADLen,
                                    const U8   * pIV,
                                    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to AES context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.3.7 Self-test API

The following table lists the AES self-test API functions.

Function	Description
<code>CRYPTO_AES_128_CBC_CAVS_SelfTest()</code>	Run CAVS AES-128 self-test.
<code>CRYPTO_AES_192_CBC_CAVS_SelfTest()</code>	Run CAVS AES-192 self-test.
<code>CRYPTO_AES_256_CBC_CAVS_SelfTest()</code>	Run CAVS AES-256 self-test.
<code>CRYPTO_AES_128_ECB_CAVS_SelfTest()</code>	Run CAVS AES-128 self-test.
<code>CRYPTO_AES_192_ECB_CAVS_SelfTest()</code>	Run CAVS AES-192 self-test.
<code>CRYPTO_AES_256_ECB_CAVS_SelfTest()</code>	Run CAVS AES-256 self-test.
<code>CRYPTO_AES_RFC3602_SelfTest()</code>	Run AES KATs from RFC 3602.
<code>CRYPTO_AES_128_CCM_CAVS_SelfTest()</code>	Run CAVS AES-128 self-test.
<code>CRYPTO_AES_192_CCM_CAVS_SelfTest()</code>	Run CAVS AES-192 self-test.
<code>CRYPTO_AES_256_CCM_CAVS_SelfTest()</code>	Run CAVS AES-256 self-test.
<code>CRYPTO_AES_128_GCM_CAVS_SelfTest()</code>	Run CAVS AES-128 self-test.
<code>CRYPTO_AES_192_GCM_CAVS_SelfTest()</code>	Run CAVS AES-192 self-test.
<code>CRYPTO_AES_256_GCM_CAVS_SelfTest()</code>	Run CAVS AES-256 self-test.
<code>CRYPTO_AES_CCM_SP800x38C_SelfTest()</code>	Run AES-CCM KATs from SP 800-38C.

7.3.7.1 CRYPTO_AES_128_CBC_CAVS_SelfTest()

Description

Run CAVS AES-128 self-test.

Prototype

```
void CRYPTO_AES_128_CBC_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.2 CRYPTO_AES_192_CBC_CAVS_SelfTest()

Description

Run CAVS AES-192 self-test.

Prototype

```
void CRYPTO_AES_192_CBC_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.3 CRYPTO_AES_256_CBC_CAVS_SelfTest()

Description

Run CAVS AES-256 self-test.

Prototype

```
void CRYPTO_AES_256_CBC_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.4 CRYPTO_AES_128_ECB_CAVS_SelfTest()

Description

Run CAVS AES-128 self-test.

Prototype

```
void CRYPTO_AES_128_ECB_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.5 CRYPTO_AES_192_ECB_CAVS_SelfTest()

Description

Run CAVS AES-192 self-test.

Prototype

```
void CRYPTO_AES_192_ECB_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.6 CRYPTO_AES_256_ECB_CAVS_SelfTest()

Description

Run CAVS AES-256 self-test.

Prototype

```
void CRYPTO_AES_256_ECB_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.7 CRYPTO_AES_128_CCM_CAVS_SelfTest()

Description

Run CAVS AES-128 self-test.

Prototype

```
void CRYPTO_AES_128_CCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.8 CRYPTO_AES_192_CCM_CAVS_SelfTest()

Description

Run CAVS AES-192 self-test.

Prototype

```
void CRYPTO_AES_192_CCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.9 CRYPTO_AES_256_CCM_CAVS_SelfTest()

Description

Run CAVS AES-256 self-test.

Prototype

```
void CRYPTO_AES_256_CCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.10 CRYPTO_AES_CCM_SP800x38C_SelfTest()

Description

Run AES-CCM KATs from SP 800-38C.

Prototype

```
void CRYPTO_AES_CCM_SP800x38C_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.11 CRYPTO_AES_128_GCM_CAVS_SelfTest()

Description

Run CAVS AES-128 self-test.

Prototype

```
void CRYPTO_AES_128_GCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.12 CRYPTO_AES_192_GCM_CAVS_SelfTest()

Description

Run CAVS AES-192 self-test.

Prototype

```
void CRYPTO_AES_192_GCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.13 CRYPTO_AES_256_GCM_CAVS_SelfTest()

Description

Run CAVS AES-256 self-test.

Prototype

```
void CRYPTO_AES_256_GCM_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.3.7.14 CRYPTO_AES_RFC3602_SelfTest()

Description

Run AES KATs from RFC 3602.

Prototype

```
void CRYPTO_AES_RFC3602_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.4 IDEA

7.4.1 Algorithm parameters

7.4.1.1 Block size

```
#define CRYPTO_IDEA_BLOCK_SIZE 8
```

The number of bytes in a single IDEA block.

7.4.1.2 Key size

```
#define CRYPTO_IDEA_KEY_SIZE 16
```

The number of bytes for the single supported key size.

7.4.2 Type-safe API

The following table lists the IDEA type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_IDEA_Install()</code>	Install cipher.
<code>CRYPTO_IDEA_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_IDEA_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_IDEA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_IDEA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_IDEA_Kill()</code>	Clear IDEA context.
Single blocks	
<code>CRYPTO_IDEA_Encrypt()</code>	Encrypt block.
<code>CRYPTO_IDEA_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_IDEA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_IDEA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_IDEA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_IDEA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_IDEA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_IDEA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_IDEA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_IDEA_CTR_Decrypt()</code>	Decrypt, CTR mode.

7.4.2.1 CRYPTO_IDEA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_IDEA_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.4.2.2 CRYPTO_IDEA_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_IDEA_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.4.2.3 CRYPTO_IDEA_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_IDEA_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                               const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.4.2.4 CRYPTO_IDEA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_IDEA_InitEncrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.4.2.5 CRYPTO_IDEA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_IDEA_InitDecrypt(      CRYPTO_IDEA_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.4.2.6 CRYPTO_IDEA_Kill()

Description

Clear IDEA context.

Prototype

```
void CRYPTO_IDEA_Kill(CRYPTO_IDEA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.4.2.7 CRYPTO_IDEA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_IDEA_Encrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                U8                  * pOutput,
                                const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.4.2.8 CRYPTO_IDEA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_IDEA_Decrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                               U8           * pOutput,
                           const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.4.2.9 CRYPTO_IDEA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_IDEA_ECB_Encrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.4.2.10 CRYPTO_IDEA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_IDEA_ECB_Decrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.4.2.11 CRYPTO_IDEA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_IDEA_CBC_Encrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.4.2.12 CRYPTO_IDEA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_IDEA_CBC_Decrypt( CRYPTO_IDEA_CONTEXT * pSelf,  
                               U8                  * pOutput,  
                               const U8            * pInput,  
                               unsigned           InputLen,  
                               U8                  * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.4.2.13 CRYPTO_IDEA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_IDEA_CTR_Encrypt( CRYPTO_IDEA_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.4.2.14 CRYPTO_IDEA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_IDEA_CTR_Decrypt( CRYPTO_IDEA_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized IDEA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.4.2.15 CRYPTO_IDEA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_IDEA_OFB_Encrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.4.2.16 CRYPTO_IDEA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_IDEA_OFB_Decrypt(          CRYPTO_IDEA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.4.3 Generic API

The following table lists the IDEA functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_IDEA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_IDEA_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_IDEA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_IDEA_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
Basic modes	
<code>CRYPTO_CIPHER_IDEA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_IDEA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_IDEA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_IDEA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_IDEA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_IDEA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_IDEA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_IDEA_CTR_0_8_Encrypt()</code>	Encrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_IDEA_CTR_4_4_Encrypt()</code>	Encrypt, CTR(4,4) mode.
<code>CRYPTO_CIPHER_IDEA_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_IDEA_CTR_0_8_Decrypt()</code>	Decrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_IDEA_CTR_4_4_Decrypt()</code>	Decrypt, CTR(4,4) mode.

7.4.3.1 CRYPTO_CIPHER_IDEA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.4.3.2 CRYPTO_CIPHER_IDEA_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_IDEA_128_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pKey	Pointer to key.

7.4.3.3 CRYPTO_CIPHER_IDEA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.4.3.4 CRYPTO_CIPHER_IDEA_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_IDEA_128_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pKey	Pointer to key.

7.4.3.5 CRYPTO_CIPHER_IDEA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_IDEA_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.4.3.6 CRYPTO_CIPHER_IDEA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_IDEA_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.4.3.7 CRYPTO_CIPHER_IDEA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_ECB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.4.3.8 CRYPTO_CIPHER_IDEA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_ECB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.4.3.9 CRYPTO_CIPHER_IDEA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CBC_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.4.3.10 CRYPTO_CIPHER_IDEA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CBC_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.4.3.11 CRYPTO_CIPHER_IDEA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_OFB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.4.3.12 CRYPTO_CIPHER_IDEA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_OFB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.4.3.13 CRYPTO_CIPHER_IDEA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_Encrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.4.3.14 CRYPTO_CIPHER_IDEA_CTR_0_8_Encrypt()

Description

Encrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_0_8_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.4.3.15 CRYPTO_CIPHER_IDEA_CTR_4_4_Encrypt()

Description

Encrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_4_4_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.4.3.16 CRYPTO_CIPHER_IDEA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_Decrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.4.3.17 CRYPTO_CIPHER_IDEA_CTR_0_8_Decrypt()

Description

Decrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_0_8_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.4.3.18 CRYPTO_CIPHER_IDEA_CTR_4_4_Decrypt()

Description

Decrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_IDEA_CTR_4_4_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to IDEA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.4.4 Self-test API

The following table lists the IDEA self-test API functions.

Function	Description
CRYPTO_IDEA_Ascom_SelfTest()	Run IDEA KATs from Ascom.

7.4.4.1 CRYPTO_IDEA_Ascom_SelfTest()

Description

Run IDEA KATs from Ascom.

Prototype

```
void CRYPTO_IDEA_Ascom_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.5 SEED

7.5.1 Standards reference

SEED is specified by the following document:

- IETF RFC 4269 — *The SEED Encryption Algorithm*

7.5.2 Algorithm parameters

7.5.2.1 Block size

```
#define CRYPTO_SEED_BLOCK_SIZE 16
```

The number of bytes in a single SEED block.

7.5.2.2 Key size

```
#define CRYPTO_SEED_KEY_SIZE 16
```

The number of bytes for the single the supported key size.

7.5.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_SEED_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize SEED to place tables in RAM rather than flash and to optimized the table sizes. Optimization levels are 0 through 3 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.14 KB	Flash	0.5 KB	0.5 KB	1.0 KB
1	0.14 KB	Flash	4.0 KB	0.4 KB	4.4 KB
2	0.14 KB	RAM	0.5 KB	0.5 KB	1.0 KB
3	0.14 KB	RAM	4.0 KB	0.4 KB	4.4 KB

7.5.4 Type-safe API

The following table lists the SEED type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_SEED_Install()</code>	Install cipher.
<code>CRYPTO_SEED_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_SEED_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_SEED_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_SEED_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_SEED_Kill()</code>	Clear SEED context.
Single blocks	
<code>CRYPTO_SEED_Encrypt()</code>	Encrypt block.
<code>CRYPTO_SEED_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_SEED_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_SEED_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_SEED_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_SEED_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_SEED_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_SEED_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_SEED_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_SEED_CTR_Decrypt()</code>	Decrypt, CTR mode.
AEAD modes	
<code>CRYPTO_SEED_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_SEED_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_SEED_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_SEED_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.5.4.1 CRYPTO_SEED_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_SEED_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.4.2 CRYPTO_SEED_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_SEED_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.5.4.3 CRYPTO_SEED_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_SEED_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                           const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.5.4.4 CRYPTO_SEED_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_SEED_InitEncrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                const U8           * pKey,
                                unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.5.4.5 CRYPTO_SEED_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_SEED_InitDecrypt(      CRYPTO_SEED_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.5.4.6 CRYPTO_SEED_Kill()

Description

Clear SEED context.

Prototype

```
void CRYPTO_SEED_Kill(CRYPTO_SEED_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.5.4.7 CRYPTO_SEED_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_SEED_Encrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                U8                  * pOutput,
                                const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.5.4.8 CRYPTO_SEED_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_SEED_Decrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                               U8           * pOutput,
                           const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.5.4.9 CRYPTO_SEED_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_SEED_ECB_Encrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                         U8                      * pOutput,
                                         const U8                  * pInput,
                                         unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.5.4.10 CRYPTO_SEED_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_SEED_ECB_Decrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.5.4.11 CRYPTO_SEED_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_SEED_CBC_Encrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                      U8           * pOutput,
const U8           * pInput,
unsigned          InputLen,
                                      U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.5.4.12 CRYPTO_SEED_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_SEED_CBC_Decrypt( CRYPTO_SEED_CONTEXT * pSelf,  
                             U8 * pOutput,  
                             const U8 * pInput,  
                             unsigned InputLen,  
                             U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.5.4.13 CRYPTO_SEED_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_SEED_CTR_Encrypt( CRYPTO_SEED_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.5.4.14 CRYPTO_SEED_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_SEED_CTR_Decrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pCTR,
                                     unsigned      CTRIndex,
                                     unsigned      CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized SEED context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.5.4.15 CRYPTO_SEED_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_SEED_OFB_Encrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                      U8           * pOutput,
const U8           * pInput,
unsigned          InputLen,
                                      U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.5.4.16 CRYPTO_SEED_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_SEED_OFB_Decrypt(          CRYPTO_SEED_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.5.4.17 CRYPTO_SEED_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_SEED_CCM_Encrypt( CRYPTO_SEED_CONTEXT * pSelf,
                               U8                  * pOutput,
                               U8                  * pTag,
                               unsigned            TagLen,
                               const U8           * pInput,
                               unsigned            InputLen,
                               const U8           * pAAD,
                               unsigned            AADLen,
                               const U8           * pIV,
                               unsigned            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.5.4.18 CRYPTO_SEED_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_SEED_CCM_Decrypt( CRYPTO_SEED_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pTag,
                             const U8 TagLen,
                             const U8 * pInput,
                             const U8 InputLen,
                             const U8 * pAAD,
                             const U8 AADLen,
                             const U8 * pIV,
                             const U8 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.5.4.19 CRYPTO_SEED_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_SEED_GCM_Encrypt( CRYPTO_SEED_CONTEXT * pSelf,
                               U8                  * pOutput,
                               U8                  * pTag,
                               unsigned            TagLen,
                               const U8           * pInput,
                               unsigned            InputLen,
                               const U8           * pAAD,
                               unsigned            AADLen,
                               const U8           * pIV,
                               unsigned            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the initialization vector.

7.5.4.20 CRYPTO_SEED_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_SEED_GCM_Decrypt( CRYPTO_SEED_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pTag,
                             U8 TagLen,
                             const U8 * pInput,
                             U8 InputLen,
                             const U8 * pAAD,
                             U8 AADLen,
                             const U8 * pIV,
                             U8 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.5.5 Generic API

The following table lists the SEED functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_SEED_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_SEED_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_SEED_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_SEED_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_SEED_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_SEED_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_SEED_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_SEED_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Basic modes	
<code>CRYPTO_CIPHER_SEED_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_SEED_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_SEED_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_SEED_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_SEED_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_SEED_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_SEED_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_SEED_CTR_0_16_Encrypt()</code>	Encrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_SEED_CTR_12_4_Encrypt()</code>	Encrypt, CTR(12,4) mode.
<code>CRYPTO_CIPHER_SEED_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_SEED_CTR_0_16_Decrypt()</code>	Decrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_SEED_CTR_12_4_Decrypt()</code>	Decrypt, CTR(12,4) mode.
AEAD modes	
<code>CRYPTO_CIPHER_SEED_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_CIPHER_SEED_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_CIPHER_SEED_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_CIPHER_SEED_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.5.5.1 CRYPTO_CIPHER_SEED_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_SEED_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.5.5.2 CRYPTO_CIPHER_SEED_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_128_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.3 CRYPTO_CIPHER_SEED_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_192_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.4 CRYPTO_CIPHER_SEED_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_256_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.5 CRYPTO_CIPHER_SEED_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_SEED_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.5.5.6 CRYPTO_CIPHER_SEED_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_128_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.7 CRYPTO_CIPHER_SEED_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_192_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.8 CRYPTO_CIPHER_SEED_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_SEED_256_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pKey	Pointer to key.

7.5.5.9 CRYPTO_CIPHER_SEED_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_SEED_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.5.5.10 CRYPTO_CIPHER_SEED_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_SEED_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.5.5.11 CRYPTO_CIPHER_SEED_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_SEED_ECB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.5.5.12 CRYPTO_CIPHER_SEED_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_SEED_ECB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.5.5.13 CRYPTO_CIPHER_SEED_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CBC_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.5.5.14 CRYPTO_CIPHER_SEED_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CBC_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.5.5.15 CRYPTO_CIPHER_SEED_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_SEED_OFB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.5.5.16 CRYPTO_CIPHER_SEED_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_SEED_OFB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.5.5.17 CRYPTO_CIPHER_SEED_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_Encrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.5.5.18 CRYPTO_CIPHER_SEED_CTR_0_16_Encrypt()

Description

Encrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_0_16_Encrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.5.5.19 CRYPTO_CIPHER_SEED_CTR_12_4_Encrypt()

Description

Encrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_12_4_Encrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.5.5.20 CRYPTO_CIPHER_SEED_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_Decrypt(      void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.5.5.21 CRYPTO_CIPHER_SEED_CTR_0_16_Decrypt()

Description

Decrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_0_16_Decrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.5.5.22 CRYPTO_CIPHER_SEED_CTR_12_4_Decrypt()

Description

Decrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CTR_12_4_Decrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.5.5.23 CRYPTO_CIPHER_SEED_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CIPHER_SEED_CCM_Encrypt( void      * pContext,
                                      U8       * pOutput,
                                      U8       * pTag,
                                      unsigned TagLen,
                                      const U8   * pInput,
                                      unsigned InputLen,
                                      const U8   * pAAD,
                                      unsigned AADLen,
                                      const U8   * pIV,
                                      unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

7.5.5.24 CRYPTO_CIPHER_SEED_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CIPHER_SEED_CCM_Decrypt( void      * pContext,
                                    U8        * pOutput,
                                    const U8   * pTag,
                                    unsigned   TagLen,
                                    const U8   * pInput,
                                    unsigned   InputLen,
                                    const U8   * pAAD,
                                    unsigned   AADLen,
                                    const U8   * pIV,
                                    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.5.5.25 CRYPTO_CIPHER_SEED_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CIPHER_SEED_GCM_Encrypt( void      * pContext,
                                      U8       * pOutput,
                                      U8       * pTag,
                                      unsigned TagLen,
                                      const U8   * pInput,
                                      unsigned InputLen,
                                      const U8   * pAAD,
                                      unsigned AADLen,
                                      const U8   * pIV,
                                      unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

7.5.5.26 CRYPTO_CIPHER_SEED_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CIPHER_SEED_GCM_Decrypt( void      * pContext,
                                    U8        * pOutput,
                                    const U8   * pTag,
                                    unsigned   TagLen,
                                    const U8   * pInput,
                                    unsigned   InputLen,
                                    const U8   * pAAD,
                                    unsigned   AADLen,
                                    const U8   * pIV,
                                    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to SEED context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.5.6 Self-test API

The following table lists the SEED self-test API functions.

Function	Description
CRYPTO_SEED_RFC4269_SelfTest()	Run SEED KATs from RFC 4269.

7.5.6.1 CRYPTO_SEED_RFC4269_SelfTest()

Description

Run SEED KATs from RFC 4269.

Prototype

```
void CRYPTO_SEED_RFC4269_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.6 ARIA

7.6.1 Standards reference

ARIA is specified by the following document:

- [IETF RFC 5794 — A Description of the ARIA Encryption Algorithm](#)

7.6.2 Algorithm parameters

7.6.2.1 Block size

```
#define CRYPTO_ARIA_BLOCK_SIZE 16
```

The number of bytes in a single ARIA block.

7.6.2.2 Key size

```
#define CRYPTO_ARIA128_KEY_SIZE 16
#define CRYPTO_ARIA192_KEY_SIZE 24
#define CRYPTO_ARIA256_KEY_SIZE 32
```

The number of bytes for each of the supported key sizes.

7.6.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_ARIA_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize ARIA to place tables in RAM rather than flash.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.28 KB	Flash	1.0 KB	1.9 KB	2.9 KB
1	0.28 KB	RAM	1.0 KB	1.9 KB	2.9 KB

7.6.4 Type-safe API

The following table lists the ARIA type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_ARIA_Install()</code>	Install cipher.
<code>CRYPTO_ARIA_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_ARIA_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_ARIA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_ARIA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_ARIA_Kill()</code>	Clear ARIA context.
Single blocks	
<code>CRYPTO_ARIA_Encrypt()</code>	Encrypt block.
<code>CRYPTO_ARIA_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_ARIA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_ARIA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_ARIA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_ARIA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_ARIA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_ARIA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_ARIA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_ARIA_CTR_Decrypt()</code>	Decrypt, CTR mode.
AEAD modes	
<code>CRYPTO_ARIA_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_ARIA_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_ARIA_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_ARIA_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.6.4.1 CRYPTO_ARIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_ARIA_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.6.4.2 CRYPTO_ARIA_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_ARIA_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.6.4.3 CRYPTO_ARIA_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_ARIA_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                               const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.6.4.4 CRYPTO_ARIA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_ARIA_InitEncrypt(      CRYPTO_ARIA_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.6.4.5 CRYPTO_ARIA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_ARIA_InitDecrypt(      CRYPTO_ARIA_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.6.4.6 CRYPTO_ARIA_Kill()

Description

Clear ARIA context.

Prototype

```
void CRYPTO_ARIA_Kill(CRYPTO_ARIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.6.4.7 CRYPTO_ARIA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_ARIA_Encrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                U8                      * pOutput,
                                const U8                  * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.6.4.8 CRYPTO_ARIA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_ARIA_Decrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                               U8           * pOutput,
                           const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.6.4.9 CRYPTO_ARIA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_ARIA_ECB_Encrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.6.4.10 CRYPTO_ARIA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_ARIA_ECB_Decrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.6.4.11 CRYPTO_ARIA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_ARIA_CBC_Encrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.6.4.12 CRYPTO_ARIA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_ARIA_CBC_Decrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned      InputLen,
                                    U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.6.4.13 CRYPTO_ARIA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_ARIA_CTR_Encrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pCTR,
                                     unsigned      CTRIndex,
                                     unsigned      CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.6.4.14 CRYPTO_ARIA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_ARIA_CTR_Decrypt( CRYPTO_ARIA_CONTEXT * pSelf,
                               U8                      * pOutput,
                               const U8                * pInput,
                               unsigned                 InputLen,
                               U8                      * pCTR,
                               unsigned                 CTRIndex,
                               unsigned                 CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized ARIA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.6.4.15 CRYPTO_ARIA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_ARIA_OFB_Encrypt(          CRYPTO_ARIA_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.6.4.16 CRYPTO_ARIA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_ARIA_OFB_Decrypt( CRYPTO_ARIA_CONTEXT * pSelf,  
                               U8                  * pOutput,  
                               const U8            * pInput,  
                               unsigned           InputLen,  
                               U8                  * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.6.4.17 CRYPTO_ARIA_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_ARIA_CCM_Encrypt( CRYPTO_ARIA_CONTEXT * pSelf,
                               U8                  * pOutput,
                               U8                  * pTag,
                               unsigned            TagLen,
                               const U8           * pInput,
                               unsigned            InputLen,
                               const U8           * pAAD,
                               unsigned            AADLen,
                               const U8           * pIV,
                               unsigned            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.6.4.18 CRYPTO_ARIA_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_ARIA_CCM_Decrypt( CRYPTO_ARIA_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pTag,
                             U8 TagLen,
                             const U8 * pInput,
                             U8 InputLen,
                             const U8 * pAAD,
                             U8 AADLen,
                             const U8 * pIV,
                             U8 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.6.4.19 CRYPTO_ARIA_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_ARIA_GCM_Encrypt( CRYPTO_ARIA_CONTEXT * pSelf,
                               U8                  * pOutput,
                               U8                  * pTag,
                               unsigned            TagLen,
                               const U8           * pInput,
                               unsigned            InputLen,
                               const U8           * pAAD,
                               unsigned            AADLen,
                               const U8           * pIV,
                               unsigned            IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the initialization vector.

7.6.4.20 CRYPTO_ARIA_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_ARIA_GCM_Decrypt( CRYPTO_ARIA_CONTEXT * pSelf,
                             U8 * pOutput,
                             const U8 * pTag,
                             TagLen,
                             const U8 * pInput,
                             InputLen,
                             const U8 * pAAD,
                             AADLen,
                             const U8 * pIV,
                             IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.6.5 Generic API

The following table lists the ARIA functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_ARIA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_ARIA_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_ARIA_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_ARIA_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_ARIA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_ARIA_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_ARIA_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_ARIA_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Basic modes	
<code>CRYPTO_CIPHER_ARIA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_ARIA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_ARIA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_ARIA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_ARIA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_ARIA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_ARIA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_ARIA_CTR_0_16_Encrypt()</code>	Encrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_ARIA_CTR_12_4_Encrypt()</code>	Encrypt, CTR(12,4) mode.
<code>CRYPTO_CIPHER_ARIA_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_ARIA_CTR_0_16_Decrypt()</code>	Decrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_ARIA_CTR_12_4_Decrypt()</code>	Decrypt, CTR(12,4) mode.
AEAD modes	
<code>CRYPTO_CIPHER_ARIA_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_CIPHER_ARIA_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_CIPHER_ARIA_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_CIPHER_ARIA_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.6.5.1 CRYPTO_CIPHER_ARIA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.6.5.2 CRYPTO_CIPHER_ARIA_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_128_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.3 CRYPTO_CIPHER_ARIA_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_192_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.4 CRYPTO_CIPHER_ARIA_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_256_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.5 CRYPTO_CIPHER_ARIA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.6.5.6 CRYPTO_CIPHER_ARIA_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_128_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.7 CRYPTO_CIPHER_ARIA_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_192_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.8 CRYPTO_CIPHER_ARIA_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_ARIA_256_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pKey	Pointer to key.

7.6.5.9 CRYPTO_CIPHER_ARIA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_ARIA_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.6.5.10 CRYPTO_CIPHER_ARIA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_ARIA_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.6.5.11 CRYPTO_CIPHER_ARIA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_ECB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.6.5.12 CRYPTO_CIPHER_ARIA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_ECB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.6.5.13 CRYPTO_CIPHER_ARIA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CBC_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.6.5.14 CRYPTO_CIPHER_ARIA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CBC_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.6.5.15 CRYPTO_CIPHER_ARIA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_OFB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.6.5.16 CRYPTO_CIPHER_ARIA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_OFB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.6.5.17 CRYPTO_CIPHER_ARIA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_Encrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.6.5.18 CRYPTO_CIPHER_ARIA_CTR_0_16_Encrypt()

Description

Encrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_0_16_Encrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.6.5.19 CRYPTO_CIPHER_ARIA_CTR_12_4_Encrypt()

Description

Encrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_12_4_Encrypt( void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned InputLen,
                                             U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.6.5.20 CRYPTO_CIPHER_ARIA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_Decrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.6.5.21 CRYPTO_CIPHER_ARIA_CTR_0_16_Decrypt()

Description

Decrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_0_16_Decrypt( void      * pContext,
                                            U8       * pOutput,
                                            const U8   * pInput,
                                            unsigned InputLen,
                                            U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.6.5.22 CRYPTO_CIPHER_ARIA_CTR_12_4_Decrypt()

Description

Decrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CTR_12_4_Decrypt( void      * pContext,
                                            U8       * pOutput,
                                            const U8   * pInput,
                                            unsigned InputLen,
                                            U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.6.5.23 CRYPTO_CIPHER_ARIA_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_CCM_Encrypt( void      * pContext,
                                      U8       * pOutput,
                                      U8       * pTag,
                                      unsigned TagLen,
                                      const U8   * pInput,
                                      unsigned InputLen,
                                      const U8   * pAAD,
                                      unsigned AADLen,
                                      const U8   * pIV,
                                      unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

7.6.5.24 CRYPTO_CIPHER_ARIA_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CIPHER_ARIA_CCM_Decrypt( void      * pContext,
                                    U8        * pOutput,
                                    const U8   * pTag,
                                    unsigned   TagLen,
                                    const U8   * pInput,
                                    unsigned   InputLen,
                                    const U8   * pAAD,
                                    unsigned   AADLen,
                                    const U8   * pIV,
                                    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.6.5.25 CRYPTO_CIPHER_ARIA_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CIPHER_ARIA_GCM_Encrypt( void      * pContext,
                                      U8       * pOutput,
                                      U8       * pTag,
                                      unsigned TagLen,
                                      const U8   * pInput,
                                      unsigned InputLen,
                                      const U8   * pAAD,
                                      unsigned AADLen,
                                      const U8   * pIV,
                                      unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

7.6.5.26 CRYPTO_CIPHER_ARIA_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CIPHER_ARIA_GCM_Decrypt( void      * pContext,
                                     U8        * pOutput,
                                     const U8   * pTag,
                                     unsigned   TagLen,
                                     const U8   * pInput,
                                     unsigned   InputLen,
                                     const U8   * pAAD,
                                     unsigned   AADLen,
                                     const U8   * pIV,
                                     unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to ARIA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.6.6 Self-test API

The following table lists the ARIA self-test API functions.

Function	Description
CRYPTO_ARIA_RFC5794_SelfTest()	Run ARIA KATs from RFC 5794.

7.6.6.1 CRYPTO_ARIA_RFC5794_SelfTest()

Description

Run ARIA KATs from RFC 5794.

Prototype

```
void CRYPTO_ARIA_RFC5794_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.7 Camellia

7.7.1 Standards reference

Camellia is specified by the following document:

- [IETF RFC 3713 — A Description of the Camellia Encryption Algorithm](#)

The CBC, CTR, and CCM modes for Camellia are defined by this document:

- [IETF RFC 5529 — Modes of Operation for Camellia for Use with IPsec](#)

7.7.2 Algorithm parameters

7.7.2.1 Block size

```
#define CRYPTO_CAMELLIA_BLOCK_SIZE 16
```

The number of bytes in a single Camellia block.

7.7.2.2 Key size

```
#define CRYPTO_CAMELLIA128_KEY_SIZE 16  
#define CRYPTO_CAMELLIA192_KEY_SIZE 24  
#define CRYPTO_CAMELLIA256_KEY_SIZE 32
```

The number of bytes for each of the supported key sizes.

7.7.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_CAMELLIA_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Camellia to use more efficient tables. Optimization levels are 0 (smallest) to 3 (fastest).

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.27 KB	Flash	1.0 KB	28.8 KB	29.8 KB
1	0.27 KB	Flash	4.0 KB	20.7 KB	24.7 KB
2	0.27 KB	RAM	1.0 KB	28.8 KB	29.8 KB
3	0.27 KB	RAM	4.0 KB	20.7 KB	24.7 KB

7.7.4 Type-safe API

The following table lists the Camellia type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_CAMELLIA_Install()</code>	Install cipher.
<code>CRYPTO_CAMELLIA_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_CAMELLIA_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_CAMELLIA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CAMELLIA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CAMELLIA_Kill()</code>	Clear Camellia context.
Single blocks	
<code>CRYPTO_CAMELLIA_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CAMELLIA_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CAMELLIA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CAMELLIA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CAMELLIA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CAMELLIA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CAMELLIA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CAMELLIA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CAMELLIA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CAMELLIA_CTR_Decrypt()</code>	Decrypt, CTR mode.
AEAD modes	
<code>CRYPTO_CAMELLIA_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_CAMELLIA_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_CAMELLIA_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_CAMELLIA_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.7.4.1 CRYPTO_CAMELLIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_CAMELLIA_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                           const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.7.4.2 CRYPTO_CAMELLIA_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_CAMELLIA_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.7.4.3 CRYPTO_CAMELLIA_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_CAMELLIA_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                                   const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.7.4.4 CRYPTO_CAMELLIA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CAMELLIA_InitEncrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  const U8 * pKey,
                                  unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.7.4.5 CRYPTO_CAMELLIA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CAMELLIA_InitDecrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  const U8 * pKey,
                                  unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.7.4.6 CRYPTO_CAMELLIA_Kill()

Description

Clear Camellia context.

Prototype

```
void CRYPTO_CAMELLIA_Kill(CRYPTO_CAMELLIA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.7.4.7 CRYPTO_CAMELLIA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CAMELLIA_Encrypt(          CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                     U8                      * pOutput,
                                     const U8                  * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.7.4.8 CRYPTO_CAMELLIA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CAMELLIA_Decrypt(          CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                    U8                  * pOutput,
                                    const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.7.4.9 CRYPTO_CAMELLIA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CAMELLIA_ECB_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.7.4.10 CRYPTO_CAMELLIA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CAMELLIA_ECB_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.7.4.11 CRYPTO_CAMELLIA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CAMELLIA_CBC_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen,  
                                  U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.7.4.12 CRYPTO_CAMELLIA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CAMELLIA_CBC_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen,  
                                  U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.7.4.13 CRYPTO_CAMELLIA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CAMELLIA_OFB_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                    U8                      * pOutput,  
                                    const U8                * pInput,  
                                    unsigned               InputLen,  
                                    U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.7.4.14 CRYPTO_CAMELLIA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CAMELLIA_OFB_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.7.4.15 CRYPTO_CAMELLIA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CAMELLIA_CTR_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  U8 * pOutput,
                                  const U8 * pInput,
                                  unsigned InputLen,
                                  U8 * pCTR,
                                  unsigned CTRIndex,
                                  unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.7.4.16 CRYPTO_CAMELLIA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CAMELLIA_CTR_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  U8 * pOutput,
                                  const U8 * pInput,
                                  unsigned InputLen,
                                  U8 * pCTR,
                                  unsigned CTRIndex,
                                  unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Camellia context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.7.4.17 CRYPTO_CAMELLIA_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CAMELLIA_CCM_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                    U8                      * pOutput,
                                    U8                      * pTag,
                                    TagLen,
                                    const U8                * pInput,
                                    InputLen,
                                    const U8                * pAAD,
                                    AADLen,
                                    const U8                * pIV,
                                    IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.7.4.18 CRYPTO_CAMELLIA_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CAMELLIA_CCM_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                 * pTag,
                                  TagLen,
                                  const U8                 * pInput,
                                  InputLen,
                                  const U8                 * pAAD,
                                  AADLen,
                                  const U8                 * pIV,
                                  unsigned                IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.7.4.19 CRYPTO_CAMELLIA_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CAMELLIA_GCM_Encrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                    U8                      * pOutput,
                                    U8                      * pTag,
                                    TagLen,
                                    const U8                * pInput,
                                    InputLen,
                                    const U8                * pAAD,
                                    AADLen,
                                    const U8                * pIV,
                                    IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to CAMELLIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the initialization vector.

7.7.4.20 CRYPTO_CAMELLIA_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CAMELLIA_GCM_Decrypt( CRYPTO_CAMELLIA_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                 * pTag,
                                  TagLen,
                                  const U8                 * pInput,
                                  InputLen,
                                  const U8                 * pAAD,
                                  AADLen,
                                  const U8                 * pIV,
                                  unsigned                IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to CAMELLIA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.7.5 Generic API

The following table lists the Camellia functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_CAMELLIA_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_CAMELLIA_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_CAMELLIA_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_CAMELLIA_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_CAMELLIA_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_CAMELLIA_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_CAMELLIA_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_CAMELLIA_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Single blocks	
<code>CRYPTO_CIPHER_CAMELLIA_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CIPHER_CAMELLIA_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CIPHER_CAMELLIA_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_CAMELLIA_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_CAMELLIA_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_CAMELLIA_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_CAMELLIA_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_CAMELLIA_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Encrypt()</code>	Encrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Decrypt()</code>	Decrypt, CTR(0,16) mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Encrypt()</code>	Encrypt, CTR(12,4) mode.
<code>CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Decrypt()</code>	Decrypt, CTR(12,4) mode.
AEAD modes	
<code>CRYPTO_CIPHER_CAMELLIA_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_CIPHER_CAMELLIA_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_CIPHER_CAMELLIA_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_CIPHER_CAMELLIA_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.7.5.1 CRYPTO_CIPHER_CAMELLIA_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_InitEncrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.7.5.2 CRYPTO_CIPHER_CAMELLIA_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_128_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.3 CRYPTO_CIPHER_CAMELLIA_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_192_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.4 CRYPTO_CIPHER_CAMELLIA_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_256_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.5 CRYPTO_CIPHER_CAMELLIA_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_InitDecrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.7.5.6 CRYPTO_CIPHER_CAMELLIA_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_128_InitDecrypt(      void * pContext,  
                                              const U8    * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.7 CRYPTO_CIPHER_CAMELLIA_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_192_InitDecrypt(      void * pContext,  
                                              const U8    * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.8 CRYPTO_CIPHER_CAMELLIA_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_256_InitDecrypt(      void * pContext,  
                                              const U8    * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pKey	Pointer to key.

7.7.5.9 CRYPTO_CIPHER_CAMELLIA_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_Encrypt(      void * pContext,
                                             U8   * pOutput,
                                         const U8   * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.7.5.10 CRYPTO_CIPHER_CAMELLIA_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.7.5.11 CRYPTO_CIPHER_CAMELLIA_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_ECB_Encrypt(      void      * pContext,
                                                U8       * pOutput,
                                                const U8   * pInput,
                                                unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.7.5.12 CRYPTO_CIPHER_CAMELLIA_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_ECB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.7.5.13 CRYPTO_CIPHER_CAMELLIA_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CBC_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.7.5.14 CRYPTO_CIPHER_CAMELLIA_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CBC_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.7.5.15 CRYPTO_CIPHER_CAMELLIA_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_OFB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.7.5.16 CRYPTO_CIPHER_CAMELLIA_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_OFB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.7.5.17 CRYPTO_CIPHER_CAMELLIA_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_Encrypt( void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.7.5.18 CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Encrypt()

Description

Encrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Encrypt(      void      * pContext,
                                                    U8       * pOutput,
const U8      * pInput,
unsigned     InputLen,
U8      * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.7.5.19 CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Encrypt()

Description

Encrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Encrypt(  
    void      * pContext,  
    U8        * pOutput,  
    const U8   * pInput,  
    unsigned   InputLen,  
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[12...15]`.

7.7.5.20 CRYPTO_CIPHER_CAMELLIA_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_Decrypt( void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.7.5.21 CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Decrypt()

Description

Decrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Decrypt(      void      * pContext,
                                                    U8       * pOutput,
const U8      * pInput,
unsigned     InputLen,
U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.7.5.22 CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Decrypt()

Description

Decrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Decrypt(      void      * pContext,
                                                    U8       * pOutput,
const U8      * pInput,
unsigned     InputLen,
U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to Camellia context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.7.5.23 CRYPTO_CIPHER_CAMELLIA_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_CCM_Encrypt(
    void      * pContext,
    U8        * pOutput,
    U8        * pTag,
    unsigned   TagLen,
    const U8   * pInput,
    unsigned   InputLen,
    const U8   * pAAD,
    unsigned   AADLen,
    const U8   * pIV,
    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.7.5.24 CRYPTO_CIPHER_CAMELLIA_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CIPHER_CAMELLIA_CCM_Decrypt( void      * pContext,
                                         U8        * pOutput,
                                         const U8   * pTag,
                                         unsigned   TagLen,
                                         const U8   * pInput,
                                         unsigned   InputLen,
                                         const U8   * pAAD,
                                         unsigned   AADLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.7.5.25 CRYPTO_CIPHER_CAMELLIA_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CIPHER_CAMELLIA_GCM_Encrypt(
    void      * pContext,
    U8        * pOutput,
    U8        * pTag,
    unsigned   TagLen,
    const U8   * pInput,
    unsigned   InputLen,
    const U8   * pAAD,
    unsigned   AADLen,
    const U8   * pIV,
    unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAMELLIA context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

7.7.5.26 CRYPTO_CIPHER_CAMELLIA_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CIPHER_CAMELLIA_GCM_Decrypt( void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pTag,
                                         unsigned TagLen,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         const U8   * pAAD,
                                         unsigned AADLen,
                                         const U8   * pIV,
                                         unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAMELLIA context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.7.6 Self-test API

The following table lists the Camellia self-test API functions.

Function	Description
<code>CRYPTO_CAMELLIA_NTT_SelfTest()</code>	Run Camellia self-tests from NTT.
<code>CRYPTO_CAMELLIA_RFC5528_SelfTest()</code>	Run RFC 5528 Camellia-CTR tests.

7.7.6.1 CRYPTO_CAMELLIA_NTT_SelfTest()

Description

Run Camellia self-tests from NTT.

Prototype

```
void CRYPTO_CAMELLIA_NTT_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.7.6.2 CRYPTO_CAMELLIA_RFC5528_SelfTest()

Description

Run RFC 5528 Camellia-CTR tests.

Prototype

```
void CRYPTO_CAMELLIA_RFC5528_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.8 CAST

7.8.1 Algorithm parameters

7.8.1.1 Block size

```
#define CRYPTO_CAST_BLOCK_SIZE 8
```

The number of bytes in a single CAST-5 block.

7.8.1.2 Key size

```
#define CRYPTO_CAST128_KEY_SIZE 16  
#define CRYPTO_CAST192_KEY_SIZE 24  
#define CRYPTO_CAST256_KEY_SIZE 32
```

The number of bytes for each of the supported key sizes.

7.8.2 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_CAST_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize CAST to place tables in RAM rather than flash. Optimization levels are 0 through 1 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.10 KB	Flash	8.0 KB	3.5 KB	11.5 KB
1	0.10 KB	RAM	8.0 KB	3.7 KB	11.7 KB

7.8.3 Type-safe API

The following table lists the CAST type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_CAST_Install()</code>	Install cipher.
<code>CRYPTO_CAST_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_CAST_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_CAST_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CAST_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CAST_Kill()</code>	Clear CAST context.
Single blocks	
<code>CRYPTO_CAST_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CAST_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CAST_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CAST_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CAST_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CAST_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CAST_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CAST_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CAST_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CAST_CTR_Decrypt()</code>	Decrypt, CTR mode.

7.8.3.1 CRYPTO_CAST_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_CAST_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.8.3.2 CRYPTO_CAST_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_CAST_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.8.3.3 CRYPTO_CAST_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_CAST_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                           const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.8.3.4 CRYPTO_CAST_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CAST_InitEncrypt(      CRYPTO_CAST_CONTEXT * pSelf,
                                    const U8           * pKey,
                                    unsigned          KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.8.3.5 CRYPTO_CAST_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CAST_InitDecrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                const U8           * pKey,
                                unsigned           KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.8.3.6 CRYPTO_CAST_Kill()

Description

Clear CAST context.

Prototype

```
void CRYPTO_CAST_Kill(CRYPTO_CAST_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.

7.8.3.7 CRYPTO_CAST_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CAST_Encrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                U8             * pOutput,
                                const U8        * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.8.3.8 CRYPTO_CAST_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CAST_Decrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                               U8           * pOutput,
                           const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.8.3.9 CRYPTO_CAST_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CAST_ECB_Decrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.8.3.10 CRYPTO_CAST_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CAST_ECB_Encrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned       InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.8.3.11 CRYPTO_CAST_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CAST_CBC_Decrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                    U8           * pOutput,
                                    const U8      * pInput,
                                    unsigned       InputLen,
                                    U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.8.3.12 CRYPTO_CAST_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CAST_CBC_Encrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned       InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.8.3.13 CRYPTO_CAST_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CAST_OFB_Decrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.8.3.14 CRYPTO_CAST_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CAST_OFB_Encrypt(          CRYPTO_CAST_CONTEXT * pSelf,
                                     U8           * pOutput,
                                     const U8      * pInput,
                                     unsigned      InputLen,
                                     U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.8.3.15 CRYPTO_CAST_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CAST_CTR_Decrypt( CRYPTO_CAST_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized CAST context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.8.3.16 CRYPTO_CAST_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CAST_CTR_Encrypt( CRYPTO_CAST_CONTEXT * pSelf,
                               U8                  * pOutput,
                               const U8            * pInput,
                               unsigned           InputLen,
                               U8                  * pCTR,
                               unsigned           CTRIndex,
                               unsigned           CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.8.4 Generic API

The following table lists the CAST functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_CAST_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_CAST_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_CAST_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_CAST_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_CAST_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_CAST_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_CAST_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_CAST_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Single blocks	
<code>CRYPTO_CIPHER_CAST_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CIPHER_CAST_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CIPHER_CAST_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_CAST_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_CAST_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_CAST_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_CAST_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_CAST_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_CAST_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_CAST_CTR_0_8_Encrypt()</code>	Encrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_CAST_CTR_4_4_Encrypt()</code>	Encrypt, CTR(4,4) mode.
<code>CRYPTO_CIPHER_CAST_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_CAST_CTR_0_8_Decrypt()</code>	Decrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_CAST_CTR_4_4_Decrypt()</code>	Decrypt, CTR(4,4) mode.

7.8.4.1 CRYPTO_CIPHER_CAST_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_CAST_InitEncrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.8.4.2 CRYPTO_CIPHER_CAST_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_128_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.3 CRYPTO_CIPHER_CAST_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_192_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.4 CRYPTO_CIPHER_CAST_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_256_InitEncrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.5 CRYPTO_CIPHER_CAST_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_CAST_InitDecrypt(      void      * pContext,
                                         const U8      * pKey,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.8.4.6 CRYPTO_CIPHER_CAST_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_128_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.7 CRYPTO_CIPHER_CAST_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_192_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.8 CRYPTO_CIPHER_CAST_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_CAST_256_InitDecrypt(      void * pContext,  
                                         const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pKey	Pointer to key.

7.8.4.9 CRYPTO_CIPHER_CAST_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_CAST_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.8.4.10 CRYPTO_CIPHER_CAST_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_CAST_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.8.4.11 CRYPTO_CIPHER_CAST_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_CAST_ECB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.8.4.12 CRYPTO_CIPHER_CAST_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_CAST_ECB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                         const U8      * pInput,
                                         unsigned     InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.8.4.13 CRYPTO_CIPHER_CAST_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CBC_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.8.4.14 CRYPTO_CIPHER_CAST_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CBC_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.8.4.15 CRYPTO_CIPHER_CAST_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_CAST_OFB_Encrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.8.4.16 CRYPTO_CIPHER_CAST_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_CAST_OFB_Decrypt(      void      * pContext,
                                             U8       * pOutput,
                                             const U8   * pInput,
                                             unsigned   InputLen,
                                             U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.8.4.17 CRYPTO_CIPHER_CAST_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_Encrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.8.4.18 CRYPTO_CIPHER_CAST_CTR_0_8_Encrypt()

Description

Encrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_0_8_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.8.4.19 CRYPTO_CIPHER_CAST_CTR_4_4_Encrypt()

Description

Encrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_4_4_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.8.4.20 CRYPTO_CIPHER_CAST_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_Decrypt(      void      * pContext,
                                           U8       * pOutput,
                                           const U8   * pInput,
                                           unsigned InputLen,
                                           U8       * pCTR,
                                           unsigned CTRIndex,
                                           unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.8.4.21 CRYPTO_CIPHER_CAST_CTR_0_8_Decrypt()

Description

Decrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_0_8_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.8.4.22 CRYPTO_CIPHER_CAST_CTR_4_4_Decrypt()

Description

Decrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_CAST_CTR_4_4_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to CAST context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.8.5 Self-test API

The following table lists the CAST self-test API functions.

Function	Description
CRYPTO_CAST_RFC2144_SelfTest()	Run CAST KATs from RFC2144.

7.8.5.1 CRYPTO_CAST_RFC2144_SelfTest()

Description

Run CAST KATs from RFC2144.

Prototype

```
void CRYPTO_CAST_RFC2144_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.9 ChaCha20

7.9.1 Algorithm parameters

7.9.1.1 Block size

```
#define CRYPTO_CHACHA20_BLOCK_SIZE 64
```

The number of bytes in a single ChaCha20 block.

7.9.1.2 Key size

```
#define CRYPTO_CHACHA20_KEY_SIZE 32
```

The number of bytes for a single supported key size.

7.9.2 Standards reference

ChaCha20 is specified by the following document:

- IETF RFC 7539 — *ChaCha20 and Poly1305 for IETF Protocols*

7.9.3 Type-safe API

The following table lists the ChaCha20 type-safe API functions.

Setup	
<code>CRYPTO_CHACHA20_InitEncrypt_32_96()</code>	Initialize cipher in encryption mode, 32-bit counter, 96-bit IV.
<code>CRYPTO_CHACHA20_InitDecrypt_32_96()</code>	Initialize cipher in decryption mode.
<code>CRYPTO_CHACHA20_InitEncrypt_64_64()</code>	Initialize cipher in encryption mode, 64-bit counter, 64-bit IV.
<code>CRYPTO_CHACHA20_InitDecrypt_64_64()</code>	Initialize cipher in decryption mode.
<code>CRYPTO_CHACHA20_SetPos()</code>	Set block position.
<code>CRYPTO_CHACHA20_SetIV()</code>	Set nonce.
<code>CRYPTO_CHACHA20_Kill()</code>	Destroy cipher context.
AEAD mode	
<code>CRYPTO_CHACHA20_POLY1305_GenKey()</code>	Generate Poly1305 key using ChaCha20.
<code>CRYPTO_CHACHA20_POLY1305_Encrypt()</code>	Encrypt, Poly1305 mode.
<code>CRYPTO_CHACHA20_POLY1305_Decrypt()</code>	Decrypt, Poly1305 mode.

7.9.3.1 CRYPTO_CHACHA20_InitEncrypt_32_96()

Description

Initialize cipher in encryption mode, 32-bit counter, 96-bit IV.

Prototype

```
void CRYPTO_CHACHA20_InitEncrypt_32_96(          CRYPTO_CHACHA20_CONTEXT * pSelf,
                                              const U8                  * pKey,
                                              unsigned                 KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
pKey	Pointer to key octet string.
KeyLen	Octet length of key octet string.

7.9.3.2 CRYPTO_CHACHA20_InitDecrypt_32_96()

Description

Initialize cipher in decryption mode.

Prototype

```
void CRYPTO_CHACHA20_InitDecrypt_32_96( CRYPTO_CHACHA20_CONTEXT * pSelf,  
                                         const U8 * pKey,  
                                         unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
pKey	Pointer to key octet string.
KeyLen	Octet length of key octet string.

7.9.3.3 CRYPTO_CHACHA20_InitEncrypt_64_64()

Description

Initialize cipher in encryption mode, 64-bit counter, 64-bit IV.

Prototype

```
void CRYPTO_CHACHA20_InitEncrypt_64_64(          CRYPTO_CHACHA20_CONTEXT * pSelf,
                                                const U8                  * pKey,
                                                unsigned                 KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
pKey	Pointer to key octet string.
KeyLen	Octet length of key octet string.

7.9.3.4 CRYPTO_CHACHA20_InitDecrypt_64_64()

Description

Initialize cipher in decryption mode.

Prototype

```
void CRYPTO_CHACHA20_InitDecrypt_64_64( CRYPTO_CHACHA20_CONTEXT * pSelf,  
                                         const U8 * pKey,  
                                         unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
pKey	Pointer to key octet string.
KeyLen	Octet length of key octet string.

7.9.3.5 CRYPTO_CHACHA20_SetPos()

Description

Set block position.

Prototype

```
void CRYPTO_CHACHA20_SetPos(CRYPTO_CHACHA20_CONTEXT * pSelf,  
                           U64 Pos);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
Pos	Block number, 32 bits in IETF mode, otherwise 64 bits.

7.9.3.6 CRYPTO_CHACHA20_SetIV()

Description

Set nonce.

Prototype

```
void CRYPTO_CHACHA20_SetIV(          CRYPTO_CHACHA20_CONTEXT * pSelf,
                                const U8           * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.
pIV	Pointer to nonce octet string, 12 octets in IETF mode, otherwise 8 octets.

7.9.3.7 CRYPTO_CHACHA20_Kill()

Description

Destroy cipher context.

Prototype

```
void CRYPTO_CHACHA20_Kill(CRYPTO_CHACHA20_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context.

7.9.3.8 CRYPTO_CHACHA20_POLY1305_GenKey()

Description

Generate Poly1305 key using ChaCha20.

Prototype

```
void CRYPTO_CHACHA20_POLY1305_GenKey(      U8 * pOutput,  
                                         const U8 * pKey,  
                                         const U8 * pNonce);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the one-time key, 32 octets.
pKey	Pointer to key octet string, 32 octets.
pNonce	Pointer to nonce octet string, 12 octets.

Additional information

The Poly1305 key is generated according to RFC 7539.

7.9.3.9 CRYPTO_CHACHA20_POLY1305_Encrypt()

Description

Encrypt, Poly1305 mode.

Prototype

```
void CRYPTO_CHACHA20_POLY1305_Encrypt( CRYPTO_CHACHA20_CONTEXT * pSelf,
                                         U8                      * pOutput,
                                         U8                      * pTag,
                                         TagLen,
                                         unsigned                * pInput,
                                         InputLen,
                                         const U8                * pAAD,
                                         AADLen,
                                         const U8                * pIV,
                                         unsigned                * IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data, 16 octets.
TagLen	Octet length of the tag, 16 octets.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector, 12 octets in IETF mode, otherwise 8 octets.

7.9.3.10 CRYPTO_CHACHA20_POLY1305_Decrypt()

Description

Decrypt, Poly1305 mode.

Prototype

```
int CRYPTO_CHACHA20_POLY1305_Decrypt( CRYPTO_CHACHA20_CONTEXT * pSelf,
                                         U8 * pOutput,
                                         const U8 * pTag,
                                         const U8 TagLen,
                                         const U8 * pInput,
                                         const U8 InputLen,
                                         const U8 * pAAD,
                                         const U8 AADLen,
                                         const U8 * pIV,
                                         const U8 IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to ChaCha20 context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector, 12 octets in IETF mode, otherwise 8 octets.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.9.4 Self-test API

The following table lists the ChaCha20 self-test API functions.

Function	Description
CRYPTO_CHACHA20_RFC7539_SelfTest()	Run ChaCha20 KATs from RFC 7539.

7.9.4.1 CRYPTO_CHACHA20_RFC7539_SelfTest()

Description

Run ChaCha20 KATs from RFC 7539.

Prototype

```
void CRYPTO_CHACHA20_RFC7539_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.10 Blowfish

7.10.1 Algorithm parameters

7.10.1.1 Block size

```
#define CRYPTO_BLOWFISH_BLOCK_SIZE 16
```

The number of bytes in a single Blowfish a block.

7.10.1.2 Key size

```
#define CRYPTO_BLOWFISH128_KEY_SIZE 16
#define CRYPTO_BLOWFISH192_KEY_SIZE 24
#define CRYPTO_BLOWFISH256_KEY_SIZE 32
```

The number of bytes for each of the supported key sizes.

7.10.2 Type-safe API

The following table lists the Blowfish type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_BLOWFISH_Install()</code>	Install cipher.
<code>CRYPTO_BLOWFISH_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_BLOWFISH_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_BLOWFISH_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_BLOWFISH_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_BLOWFISH_Kill()</code>	Clear BLOWFISH context.
Single blocks	
<code>CRYPTO_BLOWFISH_Encrypt()</code>	Encrypt block.
<code>CRYPTO_BLOWFISH_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_BLOWFISH_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_BLOWFISH_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_BLOWFISH_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_BLOWFISH_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_BLOWFISH_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_BLOWFISH_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_BLOWFISH_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_BLOWFISH_CTR_Decrypt()</code>	Decrypt, CTR mode.

7.10.2.1 CRYPTO_BLOWFISH_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_BLOWFISH_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                           const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.10.2.2 CRYPTO_BLOWFISH_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_BLOWFISH_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.10.2.3 CRYPTO_BLOWFISH_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_BLOWFISH_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                                   const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.10.2.4 CRYPTO_BLOWFISH_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_BLOWFISH_InitEncrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                  const U8 * pKey,
                                  unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.10.2.5 CRYPTO_BLOWFISH_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_BLOWFISH_InitDecrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                  const U8 * pKey,
                                  unsigned KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.10.2.6 CRYPTO_BLOWFISH_Kill()

Description

Clear BLOWFISH context.

Prototype

```
void CRYPTO_BLOWFISH_Kill(CRYPTO_BLOWFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to cipher context.

7.10.2.7 CRYPTO_BLOWFISH_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_BLOWFISH_Encrypt(          CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                     U8                  * pOutput,
                                     const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.10.2.8 CRYPTO_BLOWFISH_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_BLOWFISH_Decrypt(          CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                    U8                  * pOutput,
                                    const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.10.2.9 CRYPTO_BLOWFISH_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_BLOWFISH_ECB_Encrypt(          CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                         U8                      * pOutput,
                                         const U8                * pInput,
                                         unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.10.2.10 CRYPTO_BLOWFISH_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_BLOWFISH_ECB_Decrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.10.2.11 CRYPTO_BLOWFISH_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_BLOWFISH_CBC_Encrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.10.2.12 CRYPTO_BLOWFISH_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_BLOWFISH_CBC_Decrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.10.2.13 CRYPTO_BLOWFISH_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_BLOWFISH_OFB_Encrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.10.2.14 CRYPTO_BLOWFISH_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_BLOWFISH_OFB_Decrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.10.2.15 CRYPTO_BLOWFISH_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_BLOWFISH_CTR_Encrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                  U8 * pOutput,
                                  const U8 * pInput,
                                  unsigned InputLen,
                                  U8 * pCTR,
                                  unsigned CTRIndex,
                                  unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.10.2.16 CRYPTO_BLOWFISH_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_BLOWFISH_CTR_Decrypt( CRYPTO_BLOWFISH_CONTEXT * pSelf,
                                  U8 * pOutput,
                                  const U8 * pInput,
                                  unsigned InputLen,
                                  U8 * pCTR,
                                  unsigned CTRIndex,
                                  unsigned CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Blowfish context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.10.3 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_BLOWFISH_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Blowfish to use more efficient tables. Optimization levels are 0 to 1.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	4.0 KB	Flash	4.0 KB	0.7 KB	4.7 KB
1	4.0 KB	RAM	4.0 KB	1.1 KB	5.1 KB

7.10.4 Generic API

The following table lists the Blowfish functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_BLOWFISH_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_BLOWFISH_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_BLOWFISH_192_InitEncrypt()</code>	Initialize, encrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_BLOWFISH_256_InitEncrypt()</code>	Initialize, encrypt mode, 256-bit key.
<code>CRYPTO_CIPHER_BLOWFISH_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_BLOWFISH_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_BLOWFISH_192_InitDecrypt()</code>	Initialize, decrypt mode, 192-bit key.
<code>CRYPTO_CIPHER_BLOWFISH_256_InitDecrypt()</code>	Initialize, decrypt mode, 256-bit key.
Single blocks	
<code>CRYPTO_CIPHER_BLOWFISH_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CIPHER_BLOWFISH_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_CIPHER_BLOWFISH_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_BLOWFISH_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_BLOWFISH_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_BLOWFISH_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_BLOWFISH_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_BLOWFISH_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Encrypt()</code>	Encrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Encrypt()</code>	Encrypt, CTR(4,4) mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Decrypt()</code>	Decrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Decrypt()</code>	Decrypt, CTR(4,4) mode.

7.10.4.1 CRYPTO_CIPHER_BLOWFISH_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_InitEncrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.10.4.2 CRYPTO_CIPHER_BLOWFISH_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_128_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.3 CRYPTO_CIPHER_BLOWFISH_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_192_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.4 CRYPTO_CIPHER_BLOWFISH_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_256_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.5 CRYPTO_CIPHER_BLOWFISH_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_InitDecrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.10.4.6 CRYPTO_CIPHER_BLOWFISH_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_128_InitDecrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.7 CRYPTO_CIPHER_BLOWFISH_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_192_InitDecrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.8 CRYPTO_CIPHER_BLOWFISH_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_256_InitDecrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pKey	Pointer to key.

7.10.4.9 CRYPTO_CIPHER_BLOWFISH_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.10.4.10 CRYPTO_CIPHER_BLOWFISH_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.10.4.11 CRYPTO_CIPHER_BLOWFISH_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_ECB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.10.4.12 CRYPTO_CIPHER_BLOWFISH_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_ECB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.10.4.13 CRYPTO_CIPHER_BLOWFISH_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CBC_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.10.4.14 CRYPTO_CIPHER_BLOWFISH_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CBC_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.10.4.15 CRYPTO_CIPHER_BLOWFISH_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_OFB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.10.4.16 CRYPTO_CIPHER_BLOWFISH_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_OFB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.10.4.17 CRYPTO_CIPHER_BLOWFISH_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_Encrypt( void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.10.4.18 CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Encrypt()

Description

Encrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Encrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.10.4.19 CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Encrypt()

Description

Encrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Encrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.10.4.20 CRYPTO_CIPHER_BLOWFISH_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_Decrypt( void      * pContext,
                                         U8       * pOutput,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         U8       * pCTR,
                                         unsigned CTRIndex,
                                         unsigned CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.10.4.21 CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Decrypt()

Description

Decrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Decrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.10.4.22 CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Decrypt()

Description

Decrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Decrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to BLOWFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.10.5 Self-test API

The following table lists the Blowfish self-test API functions.

Function	Description
CRYPTO_BLOWFISH_Schneier_SelfTest()	Run Blowfish KATs from Schneier.

7.10.5.1 CRYPTO_BLOWFISH_Schneier_SelfTest()

Description

Run Blowfish KATs from Schneier.

Prototype

```
void CRYPTO_BLOWFISH_Schneier_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.11 Twofish

7.11.1 Algorithm parameters

7.11.1.1 Block size

```
#define CRYPTO_TWOFISH_BLOCK_SIZE 16
```

The number of bytes in a single Twofish block.

7.11.1.2 Key size

```
#define CRYPTO_TWOFISH128_KEY_SIZE 16  
#define CRYPTO_TWOFISH192_KEY_SIZE 24  
#define CRYPTO_TWOFISH256_KEY_SIZE 32
```

The number of bytes for each of the supported key sizes.

7.11.2 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_TWOFISH_OPTIMIZE 1
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Twofish to use more efficient tables. Optimization levels are 0 (smallest) to 15 (fastest).

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.2 KB	Flash	0.6 KB	3.4 KB	4.0 KB
1	0.2 KB	Flash	4.6 KB	3.1 KB	7.7 KB
2	0.2 KB	Flash	8.5 KB	3.2 KB	11.7 KB
3	0.2 KB	Flash	12.5 KB	2.8 KB	15.3 KB
4	4.2 KB	Flash	0.6 KB	3.4 KB	4.0 KB
5	4.2 KB	Flash	4.6 KB	3.1 KB	7.7 KB
6	4.2 KB	Flash	8.5 KB	3.2 KB	11.7 KB
7	4.2 KB	Flash	12.5 KB	2.8 KB	15.3 KB
8	0.2 KB	RAM	0.6 KB	3.4 KB	4.0 KB
9	0.2 KB	RAM	4.6 KB	3.1 KB	7.7 KB
10	0.2 KB	RAM	8.5 KB	3.2 KB	11.7 KB
11	0.2 KB	RAM	12.5 KB	2.8 KB	15.3 KB
12	4.2 KB	RAM	0.6 KB	3.4 KB	4.0 KB
13	4.2 KB	RAM	4.6 KB	3.1 KB	7.7 KB
14	4.2 KB	RAM	8.5 KB	3.2 KB	11.7 KB
15	4.2 KB	RAM	12.5 KB	2.8 KB	15.3 KB

7.11.3 Type-safe API

The following table lists the Twofish type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_TWOFISH_Install()</code>	Install cipher.
<code>CRYPTO_TWOFISH_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_TWOFISH_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_TWOFISH_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_TWOFISH_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_TWOFISH_Kill()</code>	Clear TWOFISH context.
Single blocks	
<code>CRYPTO_TWOFISH_Encrypt()</code>	Encrypt block.
<code>CRYPTO_TWOFISH_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_TWOFISH_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_TWOFISH_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_TWOFISH_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_TWOFISH_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_TWOFISH_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_TWOFISH_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_TWOFISH_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_TWOFISH_CTR_Decrypt()</code>	Decrypt, CTR mode.
AEAD modes	
<code>CRYPTO_TWOFISH_CCM_Encrypt()</code>	Encrypt, CCM mode.
<code>CRYPTO_TWOFISH_CCM_Decrypt()</code>	Decrypt, CCM mode.
<code>CRYPTO_TWOFISH_GCM_Encrypt()</code>	Encrypt, GCM mode.
<code>CRYPTO_TWOFISH_GCM_Decrypt()</code>	Decrypt, GCM mode.

7.11.3.1 CRYPTO_TWOFISH_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_TWOFISH_Install(const CRYPTO_CIPHER_API * pHWAPI,  
                           const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.11.3.2 CRYPTO_TWOFISH_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_TWOFISH_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.11.3.3 CRYPTO_TWOFISH_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_TWOFISH_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                                 const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.11.3.4 CRYPTO_TWOFISH_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_TWOFISH_InitEncrypt(      CRYPTO_TWOFISH_CONTEXT * pSelf,
                                         const U8                 * pKey,
                                         unsigned                KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.11.3.5 CRYPTO_TWOFISH_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_TWOFISH_InitDecrypt(          CRYPTO_TWOFISH_CONTEXT * pSelf,
                                    const U8                 * pKey,
                                    unsigned                  KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.11.3.6 CRYPTO_TWOFISH_Kill()

Description

Clear TWOFISH context.

Prototype

```
void CRYPTO_TWOFISH_Kill(CRYPTO_TWOFISH_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to cipher context.

7.11.3.7 CRYPTO_TWOFISH_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_TWOFISH_Encrypt(          CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8           * pOutput,
const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.11.3.8 CRYPTO_TWOFISH_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_TWOFISH_Decrypt(          CRYPTO_TWOFISH_CONTEXT * pSelf,
                                U8           * pOutput,
                                const U8      * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.11.3.9 CRYPTO_TWOFISH_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_TWOFISH_ECB_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                  * pInput,
                                  unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.11.3.10 CRYPTO_TWOFISH_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_TWOFISH_ECB_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                  * pInput,
                                  unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.11.3.11 CRYPTO_TWOFISH_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_TWOFISH_CBC_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.11.3.12 CRYPTO_TWOFISH_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_TWOFISH_CBC_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,  
                                 U8 * pOutput,  
                                 const U8 * pInput,  
                                 unsigned InputLen,  
                                 U8 * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.11.3.13 CRYPTO_TWOFISH_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_TWOFISH_OFB_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  unsigned                 InputLen,
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.11.3.14 CRYPTO_TWOFISH_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_TWOFISH_OFB_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to Twofish context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.11.3.15 CRYPTO_TWOFISH_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_TWOFISH_CTR_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  InputLen,
                                  unsigned                 * pCTR,
                                  CTRIndex,
                                  unsigned                 CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Twofish context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.11.3.16 CRYPTO_TWOFISH_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_TWOFISH_CTR_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                 * pInput,
                                  unsigned                InputLen,
                                  U8                      * pCTR,
                                  unsigned                CTRIndex,
                                  unsigned                CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized Twofish context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.11.3.17 CRYPTO_TWOFISH_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_TWOFISH_CCM_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  U8                      * pTag,
                                  TagLen,
                                  * pInput,
                                  InputLen,
                                  * pAAD,
                                  AADLen,
                                  * pIV,
                                  IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

7.11.3.18 CRYPTO_TWOFISH_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_TWOFISH_CCM_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                U8 * pOutput,
                                const U8 * pTag,
                                TagLen,
                                const U8 * pInput,
                                InputLen,
                                const U8 * pAAD,
                                AADLen,
                                const U8 * pIV,
                                IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). <code>TagLen</code> must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). <code>IVLen</code> must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.11.3.19 CRYPTO_TWOFISH_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_TWOFISH_GCM_Encrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  U8                      * pTag,
                                  TagLen,
                                  const U8                * pInput,
                                  InputLen,
                                  const U8                * pAAD,
                                  AADLen,
                                  const U8                * pIV,
                                  IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the initialization vector.

7.11.3.20 CRYPTO_TWOFISH_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_TWOFISH_GCM_Decrypt( CRYPTO_TWOFISH_CONTEXT * pSelf,
                                U8 * pOutput,
                                const U8 * pTag,
                                TagLen,
                                const U8 * pInput,
                                InputLen,
                                const U8 * pAAD,
                                AADLen,
                                const U8 * pIV,
                                IVLen);
```

Parameters

Parameter	Description
pSelf	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.11.4 Generic API

The following table lists the Twofish functions that conform to the generic cipher API.

Function	Description
Setup	
CRYPTO_CIPHER_TWOFISH_InitEncrypt()	Initialize, encrypt mode.
CRYPTO_CIPHER_TWOFISH_128_InitEncrypt()	Initialize, encrypt mode, 128-bit key.
CRYPTO_CIPHER_TWOFISH_192_InitEncrypt()	Initialize, encrypt mode, 192-bit key.
CRYPTO_CIPHER_TWOFISH_256_InitEncrypt()	Initialize, encrypt mode, 256-bit key.
CRYPTO_CIPHER_TWOFISH_InitDecrypt()	Initialize, decrypt mode.
CRYPTO_CIPHER_TWOFISH_128_InitDecrypt()	Initialize, decrypt mode, 128-bit key.
CRYPTO_CIPHER_TWOFISH_192_InitDecrypt()	Initialize, decrypt mode, 192-bit key.
CRYPTO_CIPHER_TWOFISH_256_InitDecrypt()	Initialize, decrypt mode, 256-bit key.
Basic modes	
CRYPTO_CIPHER_TWOFISH_Encrypt()	Encrypt block.
CRYPTO_CIPHER_TWOFISH_Decrypt()	Decrypt block.
CRYPTO_CIPHER_TWOFISH_ECB_Encrypt()	Encrypt, ECB mode.
CRYPTO_CIPHER_TWOFISH_ECB_Decrypt()	Decrypt, ECB mode.
CRYPTO_CIPHER_TWOFISH_CBC_Encrypt()	Encrypt, CBC mode.
CRYPTO_CIPHER_TWOFISH_CBC_Decrypt()	Decrypt, CBC mode.
CRYPTO_CIPHER_TWOFISH_OFB_Encrypt()	Encrypt, OFB mode.
CRYPTO_CIPHER_TWOFISH_OFB_Decrypt()	Decrypt, OFB mode.
CRYPTO_CIPHER_TWOFISH_CTR_Encrypt()	Encrypt, CTR mode.
CRYPTO_CIPHER_TWOFISH_CTR_0_16_Encrypt()	Encrypt, CTR(0,16) mode.
CRYPTO_CIPHER_TWOFISH_CTR_12_4_Encrypt()	Encrypt, CTR(12,4) mode.
CRYPTO_CIPHER_TWOFISH_CTR_Decrypt()	Decrypt, CTR mode.
CRYPTO_CIPHER_TWOFISH_CTR_0_16_Decrypt()	Decrypt, CTR(0,16) mode.
CRYPTO_CIPHER_TWOFISH_CTR_12_4_Decrypt()	Decrypt, CTR(12,4) mode.
AEAD modes	
CRYPTO_CIPHER_TWOFISH_CCM_Encrypt()	Encrypt, CCM mode.
CRYPTO_CIPHER_TWOFISH_CCM_Decrypt()	Decrypt, CCM mode.
CRYPTO_CIPHER_TWOFISH_GCM_Encrypt()	Encrypt, GCM mode.
CRYPTO_CIPHER_TWOFISH_GCM_Decrypt()	Decrypt, GCM mode.

7.11.4.1 CRYPTO_CIPHER_TWOFISH_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_InitEncrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.11.4.2 CRYPTO_CIPHER_TWOFISH_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_128_InitEncrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.3 CRYPTO_CIPHER_TWOFISH_192_InitEncrypt()

Description

Initialize, encrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_192_InitEncrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.4 CRYPTO_CIPHER_TWOFISH_256_InitEncrypt()

Description

Initialize, encrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_256_InitEncrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.5 CRYPTO_CIPHER_TWOFISH_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_InitDecrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned     KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.11.4.6 CRYPTO_CIPHER_TWOFISH_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_128_InitDecrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.7 CRYPTO_CIPHER_TWOFISH_192_InitDecrypt()

Description

Initialize, decrypt mode, 192-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_192_InitDecrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.8 CRYPTO_CIPHER_TWOFISH_256_InitDecrypt()

Description

Initialize, decrypt mode, 256-bit key.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_256_InitDecrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pKey	Pointer to key.

7.11.4.9 CRYPTO_CIPHER_TWOFISH_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_Encrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.11.4.10 CRYPTO_CIPHER_TWOFISH_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_Decrypt(      void * pContext,
                                         U8   * pOutput,
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.11.4.11 CRYPTO_CIPHER_TWOFISH_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_ECB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.11.4.12 CRYPTO_CIPHER_TWOFISH_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_ECB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.11.4.13 CRYPTO_CIPHER_TWOFISH_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CBC_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.11.4.14 CRYPTO_CIPHER_TWOFISH_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CBC_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.11.4.15 CRYPTO_CIPHER_TWOFISH_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_OFB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.11.4.16 CRYPTO_CIPHER_TWOFISH_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_OFB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.11.4.17 CRYPTO_CIPHER_TWOFISH_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pCTR,
                                              unsigned   CTRIndex,
                                              unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.11.4.18 CRYPTO_CIPHER_TWOFISH_CTR_0_16_Encrypt()

Description

Encrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_0_16_Encrypt(  
    void * pContext,  
    U8 * pOutput,  
    const U8 * pInput,  
    unsigned InputLen,  
    U8 * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[0...15].

7.11.4.19 CRYPTO_CIPHER_TWOFISH_CTR_12_4_Encrypt()

Description

Encrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_12_4_Encrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[12...15]`.

7.11.4.20 CRYPTO_CIPHER_TWOFISH_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pCTR,
                                              unsigned   CTRIndex,
                                              unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.11.4.21 CRYPTO_CIPHER_TWOFISH_CTR_0_16_Decrypt()

Description

Decrypt, CTR(0,16) mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_0_16_Decrypt(
    void      * pContext,
    U8        * pOutput,
    const U8   * pInput,
    unsigned   InputLen,
    U8        * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...15]`.

7.11.4.22 CRYPTO_CIPHER_TWOFISH_CTR_12_4_Decrypt()

Description

Decrypt, CTR(12,4) mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CTR_12_4_Decrypt(  
    void * pContext,  
    U8 * pOutput,  
    const U8 * pInput,  
    unsigned InputLen,  
    U8 * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes pCTR[12...15].

7.11.4.23 CRYPTO_CIPHER_TWOFISH_CCM_Encrypt()

Description

Encrypt, CCM mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_CCM_Encrypt( void      * pContext,
                                         U8       * pOutput,
                                         U8       * pTag,
                                         unsigned TagLen,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         const U8   * pAAD,
                                         unsigned AADLen,
                                         const U8   * pIV,
                                         unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of the data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector for encryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

7.11.4.24 CRYPTO_CIPHER_TWOFISH_CCM_Decrypt()

Description

Decrypt, CCM mode.

Prototype

```
int CRYPTO_CIPHER_TWOFISH_CCM_Decrypt( void      * pContext,
                                         U8        * pOutput,
                                         const U8   * pTag,
                                         unsigned   TagLen,
                                         const U8   * pInput,
                                         unsigned   InputLen,
                                         const U8   * pAAD,
                                         unsigned   AADLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to encrypted data.
InputLen	Octet length of encrypted data.
pAAD	Pointer to additional data authenticated by tag but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector for decryption.
IVLen	Octet length of the nonce (IV). IVLen must be between 7 and 13 inclusive.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.11.4.25 CRYPTO_CIPHER_TWOFISH_GCM_Encrypt()

Description

Encrypt, GCM mode.

Prototype

```
void CRYPTO_CIPHER_TWOFISH_GCM_Encrypt( void      * pContext,
                                         U8       * pOutput,
                                         U8       * pTag,
                                         unsigned TagLen,
                                         const U8   * pInput,
                                         unsigned InputLen,
                                         const U8   * pAAD,
                                         unsigned AADLen,
                                         const U8   * pIV,
                                         unsigned IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pTag	Pointer to object that receives the authentication tag calculated over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to data to be encrypted.
InputLen	Octet length of data to be encrypted.
pAAD	Pointer to additional data to be authenticated but not encrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

7.11.4.26 CRYPTO_CIPHER_TWOFISH_GCM_Decrypt()

Description

Decrypt, GCM mode.

Prototype

```
int CRYPTO_CIPHER_TWOFISH_GCM_Decrypt( void      * pContext,
                                         U8        * pOutput,
                                         const U8   * pTag,
                                         unsigned   TagLen,
                                         const U8   * pInput,
                                         unsigned   InputLen,
                                         const U8   * pAAD,
                                         unsigned   AADLen,
                                         const U8   * pIV,
                                         unsigned   IVLen);
```

Parameters

Parameter	Description
pContext	Pointer to TWOFISH context, encrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pTag	Pointer to authentication tag to verify over encrypted and additional data.
TagLen	Octet length of the tag.
pInput	Pointer to encrypted input.
InputLen	Octet length of encrypted input.
pAAD	Pointer to additional data to be authenticated but not decrypted.
AADLen	Octet length of additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.

Return value

- = 0 Calculated tag and given tag are identical.
- ≠ 0 Calculated tag and given tag are not identical.

7.11.5 Self-test API

The following table lists the Twofish self-test API functions.

Function	Description
CRYPTO_TWOFISH_Schneier_SelfTest()	Run Twofish KATs from Schneier.

7.11.5.1 CRYPTO_TWOFISH_Schneier_SelfTest()

Description

Run Twofish KATs from Schneier.

Prototype

```
void CRYPTO_TWOFISH_Schneier_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.12 PRESENT

7.12.1 Algorithm parameters

7.12.1.1 Block size

```
#define CRYPTO_PRESENT_BLOCK_SIZE 8
```

The number of bytes in a single PRESENT block.

7.12.1.2 Key size

```
#define CRYPTO_PRESENT80_KEY_SIZE 10  
#define CRYPTO_PRESENT128_KEY_SIZE 16
```

The number of bytes for each of the supported key sizes.

7.12.2 Configuration and resource use

Default

```
#define CRYPTO_CONFIG_PRESENT_OPTIMIZE 0
```

Override

To define a non-default value, define this symbol in CRYPTO_Conf.h.

Description

Set this preprocessor symbol nonzero to optimize PRESENT to place tables in RAM rather than flash.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.26 KB	Flash	0.1 KB	0.7 KB	0.8 KB
1	0.26 KB	RAM	0.1 KB	0.7 KB	0.8 KB

7.12.3 Type-safe API

The following table lists the PRESENT type-safe API functions.

Function	Description
Installation and hardware assist	
<code>CRYPTO_PRESENT_Install()</code>	Install cipher.
<code>CRYPTO_PRESENT_IsInstalled()</code>	Query whether cipher is installed.
<code>CRYPTO_PRESENT_QueryInstall()</code>	Query installed cipher.
Setup	
<code>CRYPTO_PRESENT_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_PRESENT_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_PRESENT_Kill()</code>	Clear PRESENT context.
Single blocks	
<code>CRYPTO_PRESENT_Encrypt()</code>	Encrypt block.
<code>CRYPTO_PRESENT_Decrypt()</code>	Decrypt block.
Basic modes	
<code>CRYPTO_PRESENT_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_PRESENT_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_PRESENT_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_PRESENT_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_PRESENT_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_PRESENT_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_PRESENT_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_PRESENT_CTR_Decrypt()</code>	Decrypt, CTR mode.

7.12.3.1 CRYPTO_PRESENT_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_PRESENT_Install(const CRYPTO_CIPHER_API * pHwapi,  
                           const CRYPTO_CIPHER_API * pSwapi);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.12.3.2 CRYPTO_PRESENT_IsInstalled()

Description

Query whether cipher is installed.

Prototype

```
int CRYPTO_PRESENT_IsInstalled(void);
```

Return value

= 0	Cipher is not installed.
≠ 0	Cipher is installed.

7.12.3.3 CRYPTO_PRESENT_QueryInstall()

Description

Query installed cipher.

Prototype

```
void CRYPTO_PRESENT_QueryInstall(const CRYPTO_CIPHER_API ** ppHWAPI,  
                                 const CRYPTO_CIPHER_API ** ppSWAPI);
```

Parameters

Parameter	Description
ppHWAPI	Pointer to object that receives the pointer to the preferred API.
ppSWAPI	Pointer to object that receives the pointer to the fallback API.

7.12.3.4 CRYPTO_PRESENT_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_PRESENT_InitEncrypt(          CRYPTO_PRESENT_CONTEXT * pSelf,
                                     const U8                 * pKey,
                                     unsigned                KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.12.3.5 CRYPTO_PRESENT_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_PRESENT_InitDecrypt(          CRYPTO_PRESENT_CONTEXT * pSelf,
                                     const U8                 * pKey,
                                     unsigned                KeyLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.12.3.6 CRYPTO_PRESENT_Kill()

Description

Clear PRESENT context.

Prototype

```
void CRYPTO_PRESENT_Kill(CRYPTO_PRESENT_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to cipher context.

7.12.3.7 CRYPTO_PRESENT_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_PRESENT_Encrypt(          CRYPTO_PRESENT_CONTEXT * pSelf,
                                    U8                 * pOutput,
                                    const U8            * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.12.3.8 CRYPTO_PRESENT_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_PRESENT_Decrypt(          CRYPTO_PRESENT_CONTEXT * pSelf,
                                  U8           * pOutput,
const U8           * pInput);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.12.3.9 CRYPTO_PRESENT_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_PRESENT_ECB_Encrypt( CRYPTO_PRESENT_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.12.3.10 CRYPTO_PRESENT_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_PRESENT_ECB_Decrypt( CRYPTO_PRESENT_CONTEXT * pSelf,  
                                  U8 * pOutput,  
                                  const U8 * pInput,  
                                  unsigned InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.12.3.11 CRYPTO_PRESENT_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_PRESENT_CBC_Encrypt( CRYPTO_PRESENT_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.12.3.12 CRYPTO_PRESENT_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_PRESENT_CBC_Decrypt( CRYPTO_PRESENT_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  const U8                * pInput,  
                                  unsigned               InputLen,  
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.12.3.13 CRYPTO_PRESENT_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_PRESENT_OFB_Encrypt( CRYPTO_PRESENT_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  unsigned                 InputLen,
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.12.3.14 CRYPTO_PRESENT_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_PRESENT_OFB_Decrypt( CRYPTO_PRESENT_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  unsigned                 InputLen,
                                  U8                      * pIV);
```

Parameters

Parameter	Description
pSelf	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to initialization vector.

7.12.3.15 CRYPTO_PRESENT_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_PRESENT_CTR_Encrypt( CRYPTO_PRESENT_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  unsigned                 InputLen,
                                  U8                      * pCTR,
                                  unsigned                 CTRIndex,
                                  unsigned                 CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.12.3.16 CRYPTO_PRESENT_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_PRESENT_CTR_Decrypt( CRYPTO_PRESENT_CONTEXT * pSelf,
                                  U8                      * pOutput,
                                  const U8                * pInput,
                                  unsigned                 InputLen,
                                  U8                      * pCTR,
                                  unsigned                 CTRIndex,
                                  unsigned                 CTRLen);
```

Parameters

Parameter	Description
pSelf	Initialized PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

7.12.4 Generic API

The following table lists the PRESENT functions that conform to the generic cipher API.

Function	Description
Setup	
<code>CRYPTO_CIPHER_PRESENT_InitEncrypt()</code>	Initialize, encrypt mode.
<code>CRYPTO_CIPHER_PRESENT_80_InitEncrypt()</code>	Initialize, encrypt mode, 80-bit key.
<code>CRYPTO_CIPHER_PRESENT_128_InitEncrypt()</code>	Initialize, encrypt mode, 128-bit key.
<code>CRYPTO_CIPHER_PRESENT_InitDecrypt()</code>	Initialize, decrypt mode.
<code>CRYPTO_CIPHER_PRESENT_80_InitDecrypt()</code>	Initialize, decrypt mode, 80-bit key.
<code>CRYPTO_CIPHER_PRESENT_128_InitDecrypt()</code>	Initialize, decrypt mode, 128-bit key.
Basic modes	
<code>CRYPTO_CIPHER_PRESENT_Encrypt()</code>	Encrypt block.
<code>CRYPTO_CIPHER_PRESENT_Decrypt()</code>	Decrypt block.
<code>CRYPTO_CIPHER_PRESENT_ECB_Encrypt()</code>	Encrypt, ECB mode.
<code>CRYPTO_CIPHER_PRESENT_ECB_Decrypt()</code>	Decrypt, ECB mode.
<code>CRYPTO_CIPHER_PRESENT_CBC_Encrypt()</code>	Encrypt, CBC mode.
<code>CRYPTO_CIPHER_PRESENT_CBC_Decrypt()</code>	Decrypt, CBC mode.
<code>CRYPTO_CIPHER_PRESENT_OFB_Encrypt()</code>	Encrypt, OFB mode.
<code>CRYPTO_CIPHER_PRESENT_OFB_Decrypt()</code>	Decrypt, OFB mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_Encrypt()</code>	Encrypt, CTR mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_0_8_Encrypt()</code>	Encrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_4_4_Encrypt()</code>	Encrypt, CTR(4,4) mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_Decrypt()</code>	Decrypt, CTR mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_0_8_Decrypt()</code>	Decrypt, CTR(0,8) mode.
<code>CRYPTO_CIPHER_PRESENT_CTR_4_4_Decrypt()</code>	Decrypt, CTR(4,4) mode.

7.12.4.1 CRYPTO_CIPHER_PRESENT_InitEncrypt()

Description

Initialize, encrypt mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_InitEncrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned      KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.12.4.2 CRYPTO_CIPHER_PRESENT_80_InitEncrypt()

Description

Initialize, encrypt mode, 80-bit key.

Prototype

```
void CRYPTO_CIPHER_PRESENT_80_InitEncrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.

7.12.4.3 CRYPTO_CIPHER_PRESENT_128_InitEncrypt()

Description

Initialize, encrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_PRESENT_128_InitEncrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.

7.12.4.4 CRYPTO_CIPHER_PRESENT_InitDecrypt()

Description

Initialize, decrypt mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_InitDecrypt(      void      * pContext,
                                              const U8      * pKey,
                                              unsigned      KeyLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.
KeyLen	Octet length of the key.

7.12.4.5 CRYPTO_CIPHER_PRESENT_80_InitDecrypt()

Description

Initialize, decrypt mode, 80-bit key.

Prototype

```
void CRYPTO_CIPHER_PRESENT_80_InitDecrypt(      void * pContext,  
                                              const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.

7.12.4.6 CRYPTO_CIPHER_PRESENT_128_InitDecrypt()

Description

Initialize, decrypt mode, 128-bit key.

Prototype

```
void CRYPTO_CIPHER_PRESENT_128_InitDecrypt(      void * pContext,
                                                const U8   * pKey);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pKey	Pointer to key.

7.12.4.7 CRYPTO_CIPHER_PRESENT_Encrypt()

Description

Encrypt block.

Prototype

```
void CRYPTO_CIPHER_PRESENT_Encrypt(      void * pContext,  
                                         U8   * pOutput,  
                                         const U8  * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.

7.12.4.8 CRYPTO_CIPHER_PRESENT_Decrypt()

Description

Decrypt block.

Prototype

```
void CRYPTO_CIPHER_PRESENT_Decrypt(      void * pContext,  
                                         U8   * pOutput,  
                                         const U8  * pInput);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.

7.12.4.9 CRYPTO_CIPHER_PRESENT_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_ECB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.

7.12.4.10 CRYPTO_CIPHER_PRESENT_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_ECB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned    InputLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.

7.12.4.11 CRYPTO_CIPHER_PRESENT_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CBC_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.12.4.12 CRYPTO_CIPHER_PRESENT_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CBC_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.12.4.13 CRYPTO_CIPHER_PRESENT_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_OFB_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.12.4.14 CRYPTO_CIPHER_PRESENT_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_OFB_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pIV);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, decrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.

7.12.4.15 CRYPTO_CIPHER_PRESENT_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_Encrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pCTR,
                                              unsigned   CTRIndex,
                                              unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.12.4.16 CRYPTO_CIPHER_PRESENT_CTR_0_8_Encrypt()

Description

Encrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_0_8_Encrypt(      void      * pContext,
                                                U8       * pOutput,
                                                const U8   * pInput,
                                                unsigned InputLen,
                                                U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.12.4.17 CRYPTO_CIPHER_PRESENT_CTR_4_4_Encrypt()

Description

Encrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_4_4_Encrypt(      void      * pContext,
                                                U8       * pOutput,
                                                const U8   * pInput,
                                                unsigned InputLen,
                                                U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.12.4.18 CRYPTO_CIPHER_PRESENT_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_Decrypt(      void      * pContext,
                                              U8       * pOutput,
                                              const U8   * pInput,
                                              unsigned   InputLen,
                                              U8       * pCTR,
                                              unsigned   CTRIndex,
                                              unsigned   CTRLen);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter. On return, the counter is updated such that additional blocks can be decrypted in CTR mode.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.

Additional information

The counter value covers the bytes `pCTR[CTRIndex...CTRIndex+CTRLen-1]`.

7.12.4.19 CRYPTO_CIPHER_PRESENT_CTR_0_8_Decrypt()

Description

Decrypt, CTR(0,8) mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_0_8_Decrypt(      void      * pContext,
                                                U8       * pOutput,
                                                const U8   * pInput,
                                                unsigned InputLen,
                                                U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the nonce and counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[0...7]`.

7.12.4.20 CRYPTO_CIPHER_PRESENT_CTR_4_4_Decrypt()

Description

Decrypt, CTR(4,4) mode.

Prototype

```
void CRYPTO_CIPHER_PRESENT_CTR_4_4_Decrypt(      void      * pContext,
                                                U8       * pOutput,
                                                const U8   * pInput,
                                                unsigned InputLen,
                                                U8       * pCTR);
```

Parameters

Parameter	Description
pContext	Pointer to PRESENT context, encrypt mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to an object that contains the counter. On return, the counter is updated such that additional blocks can be encrypted in CTR mode.

Additional information

The counter value covers the bytes `pCTR[4...7]`.

7.12.5 Self-test API

The following table lists the PRESENT self-test API functions.

Function	Description
<code>CRYPTO_PRESENT_CHES2007_SelfTest()</code>	Run all PRESENT KAT vectors defined by CHES 2007 PRESENT paper.

7.12.5.1 CRYPTO_PRESENT_CHES2007_SelfTest()

Description

Run all PRESENT KAT vectors defined by CHES 2007 PRESENT paper.

Prototype

```
void CRYPTO_PRESENT_CHES2007_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

7.13 RC4

7.13.1 Algorithm parameters

7.13.1.1 Key size

```
#define CRYPTO_RC4_40_KEY_SIZE      5
#define CRYPTO_RC4_128_KEY_SIZE     16
#define CRYPTO_RC4_256_KEY_SIZE     32
```

The number of bytes for each of the supported key sizes.

7.13.2 Type-safe API

The following table lists the Twofish type-safe API functions.

Function	Description
Setup	
CRYPTO_RC4_Prepares()	Prepare cipher with key.
Stream cipher	
CRYPTO_RC4_Encrypts()	Encrypt input to output using current state.
CRYPTO_RC4_Decrypts()	Decrypt input to output using current state.

7.13.2.1 CRYPTO_RC4_Encrypt()

Description

Encrypt input to output using current state.

Prototype

```
void CRYPTO_RC4_Encrypt( CRYPTO_RC4_CONTEXT * pContext,  
                          U8 * pOutput,  
                          const U8 * pInput,  
                          unsigned ByteCnt );
```

Parameters

Parameter	Description
pContext	Context for cipher.
pOutput	Output data.
pInput	Input data.
ByteCnt	Size of input and output data in bytes.

7.13.2.2 CRYPTO_RC4_Decrypt()

Description

Decrypt input to output using current state.

Prototype

```
void CRYPTO_RC4_Decrypt(      CRYPTO_RC4_CONTEXT * pContext,
                               U8                  * pOutput,
                           const U8                * pInput,
                           unsigned             ByteCnt);
```

Parameters

Parameter	Description
pContext	Context for cipher.
pOutput	Output data.
pInput	Input data.
ByteCnt	Size of input and output data in bytes.

7.13.2.3 CRYPTO_RC4_Prepares()

Description

Prepare cipher with key.

Prototype

```
void CRYPTO_RC4_Prepares(      CRYPTO_RC4_CONTEXT * pContext,
                               const U8           * pKey,
                               unsigned           KeyByteCnt);
```

Parameters

Parameter	Description
pContext	Context to prepare.
pKey	Encryption/Decryption key.
KeyByteCnt	Size of encryption key in bytes.

7.14 Building blocks

7.14.1 Cipher mode API

The following table lists the generic cipher mode API API functions.

Function	Description
Standard modes	
CRYPTO_CIPHER_ECB_Encrypt()	Encrypt, ECB mode.
CRYPTO_CIPHER_ECB_Decrypt()	Decrypt, ECB mode.
CRYPTO_CIPHER_CBC_Encrypt()	Encrypt, CBC mode.
CRYPTO_CIPHER_CBC_Decrypt()	Decrypt, CBC mode.
CRYPTO_CIPHER_OFB_Encrypt()	Encrypt, OFB mode.
CRYPTO_CIPHER_OFB_Decrypt()	Decrypt, OFB mode.
CRYPTO_CIPHER_CTR_Encrypt()	Encrypt, CTR mode.
CRYPTO_CIPHER_CTR_Decrypt()	Decrypt, CTR mode.
AEAD modes	
CRYPTO_CIPHER_CCM_Cipher()	Cipher, CCM mode.
CRYPTO_CIPHER_GCM_Cipher()	Cipher, GCM mode.
Building blocks	
CRYPTO_CIPHER_GCM_GF128_Multiply()	Multiply in GF(2^8) field.
CRYPTO_CIPHER_GCM_Plain_Cipher()	Cipher, GCM mode, bit-by-bit multiply.
CRYPTO_CIPHER_GCM_Shoup_8b_Cipher()	Cipher, GCM mode, Shoup 8-bit tables.

7.14.1.1 CRYPTO_CIPHER_ECB_Encrypt()

Description

Encrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_ECB_Encrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted data.
pInput	Pointer to object that contains the decrypted data.
InputLen	Octet length of the input and output.
pAPI	Pointer to cipher API.

7.14.1.2 CRYPTO_CIPHER_ECB_Decrypt()

Description

Decrypt, ECB mode.

Prototype

```
void CRYPTO_CIPHER_ECB_Decrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the decrypted data.
pInput	Pointer to object that contains the encrypted data.
InputLen	Octet length of the input and output.
pAPI	Pointer to cipher API.

7.14.1.3 CRYPTO_CIPHER_CBC_Encrypt()

Description

Encrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CBC_Encrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      U8           * pIV,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.
pAPI	Pointer to cipher API.

7.14.1.4 CRYPTO_CIPHER_CBC_Decrypt()

Description

Decrypt, CBC mode.

Prototype

```
void CRYPTO_CIPHER_CBC_Decrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      U8           * pIV,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, decrypt mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pIV	Pointer to object that contains the initialization vector.
pAPI	Pointer to cipher API.

7.14.1.5 CRYPTO_CIPHER_OFB_Encrypt()

Description

Encrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_OFB_Encrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      U8           * pIV,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypts mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and input.
pIV	Pointer to object containing the initialization vector.
pAPI	Pointer to cipher API.

7.14.1.6 CRYPTO_CIPHER_OFB_Decrypt()

Description

Decrypt, OFB mode.

Prototype

```
void CRYPTO_CIPHER_OFB_Decrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8     * pInput,
                                      unsigned      InputLen,
                                      U8           * pIV,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypts mode.
pOutput	Pointer to object that receives the plaintext output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and input.
pIV	Pointer to object containing the initialization vector.
pAPI	Pointer to cipher API.

7.14.1.7 CRYPTO_CIPHER_CTR_Encrypt()

Description

Encrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CTR_Encrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8    * pInput,
                                      unsigned     InputLen,
                                      U8           * pCTR,
                                      unsigned     CTRIndex,
                                      unsigned     CTRLen,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context, encrypt mode.
pOutput	Pointer to object that receives the encrypted output.
pInput	Pointer to object that contains the plaintext input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.
pAPI	Pointer to cipher API.

Additional information

Decryption in counter mode is identical to encryption as the cipher produces a keystream: the keystream is exclusive-or'd with the plaintext to produce the ciphertext and exclusive-or'd with the ciphertext to produce the plaintext.

7.14.1.8 CRYPTO_CIPHER_CTR_Decrypt()

Description

Decrypt, CTR mode.

Prototype

```
void CRYPTO_CIPHER_CTR_Decrypt(      void          * pContext,
                                      U8           * pOutput,
                                      const U8    * pInput,
                                      unsigned     InputLen,
                                      U8           * pCTR,
                                      unsigned     CTRIndex,
                                      unsigned     CTRLen,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context initialized in decryption mode.
pOutput	Pointer to object that receives the decrypted output.
pInput	Pointer to object that contains the encrypted input.
InputLen	Octet length of the input and output.
pCTR	Pointer to initialization vector with counter.
CTRIndex	Index of the first byte of the counter within the IV.
CTRLen	Octet length of the counter.
pAPI	Pointer to cipher API.

Additional information

Decryption in counter mode is identical to encryption as the cipher produces a keystream: the keystream is exclusive-or'd with the plaintext to produce the ciphertext and exclusive-or'd with the ciphertext to produce the plaintext.

7.14.1.9 CRYPTO_CIPHER_CCM_Cipher()

Description

Cipher, CCM mode.

Prototype

```
void CRYPTO_CIPHER_CCM_Cipher( void * pSelf,
                               U8      * pOutput,
                               U8      * pTag,
                               TagLen,
                               const U8   * pInput,
                               InputLen,
                               const U8   * pAAD,
                               AADLen,
                               const U8   * pIV,
                               IVLen,
                               int      Encrypt,
                               const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pSelf	Pointer to cipher context in encryption mode.
pOutput	Encrypted or decrypted data, according to Mode, size is InputLen bytes.
pTag	Pointer to object that receives the authentication tag calculated over data.
TagLen	Octet length of the authentication tag (MAC). TagLen must be 4, 6, 8, 10, 12, 14, or 16.
pInput	Pointer to data to be ciphered.
InputLen	Octet length of the data to be ciphered.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Initialization vector for encryption or decryption.
IVLen	Octet length of the nonce (IV). NLen must be between 7 and 13 inclusive.
Encrypt	Flag — nonzero for encryption, zero for decryption.
pAPI	Pointer to CIPHER API for ciphering.

7.14.1.10 CRYPTO_CIPHER_GCM_Cipher()

Description

Cipher, GCM mode.

Prototype

```
void CRYPTO_CIPHER_GCM_Cipher(          void * pContext,
                                       U8   * pOutput,
                                       U8   * pTag,
                                       unsigned TagLen,
                                       const U8 * pInput,
                                       const U8 * pAAD,
                                       const U8 * pIV,
                                       int    Encrypt,
                                       const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context initialized in encryption mode.
pOutput	Encrypted or decrypted data, according to Mode, size is InputLen bytes.
pTag	Pointer to object that receives the tag calculated over data.
TagLen	Octet length of the authentication tag.
pInput	Data to be encrypted or decrypted.
InputLen	Octet length of the input data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
Encrypt	Nonzero for encryption, zero for decryption.
pAPI	Pointer to CIPHER API for ciphering.

7.14.1.11 CRYPTO_CIPHER_GCM_GF128_Multiply()

Description

Multiply in GF(2⁸) field.

Prototype

```
void CRYPTO_CIPHER_GCM_GF128_Multiply(      U8 * pZ,  
                                              const U8 * pX,  
                                              const U8 * pH);
```

Parameters

Parameter	Description
pZ	Pointer to output block, may be identical to X.
pX	Pointer to operand #1, usually variable.
pH	Pointer to operand #2, usually fixed.

Additional information

pZ and pX may point to the same array for in-place multiplication, but pZ and pH must be distinct arrays.

7.14.1.12 CRYPTO_CIPHER_GCM_Plain_Cipher()

Description

Cipher, GCM mode, bit-by-bit multiply.

Prototype

```
void CRYPTO_CIPHER_GCM_Plain_Cipher( void * pContext,
                                      U8      U8          * pOutput,
                                      U8      unsigned     * pTag,
                                      const U8      TagLen,
                                      const U8      unsigned     * pInput,
                                      const U8      InputLen,
                                      const U8      unsigned     * pAAD,
                                      const U8      AADLen,
                                      const U8      unsigned     * pIV,
                                      const U8      IVLen,
                                      int           Encrypt,
                                      const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context initialized in encryption mode.
pOutput	Encrypted or decrypted data, according to Mode, size is InputLen bytes.
pTag	Pointer to object that receives the tag calculated over data.
TagLen	Octet length of the authentication tag.
pInput	Data to be encrypted or decrypted.
InputLen	Octet length of the input data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
Encrypt	Nonzero for encryption, zero for decryption.
pAPI	Pointer to CIPHER API for ciphering.

7.14.1.13 CRYPTO_CIPHER_GCM_Shoup_8b_Cipher()

Description

Cipher, GCM mode, Shoup 8-bit tables.

Prototype

```
void CRYPTO_CIPHER_GCM_Shoup_8b_Cipher( void * pContext,
                                         U8   * pOutput,
                                         U8   * pTag,
                                         TagLen,
                                         const U8 * pInput,
                                         InputLen,
                                         const U8 * pAAD,
                                         AADLen,
                                         const U8 * pIV,
                                         IVLen,
                                         int   Encrypt,
                                         const CRYPTO_CIPHER_API * pAPI);
```

Parameters

Parameter	Description
pContext	Pointer to cipher context initialized in encryption mode.
pOutput	Encrypted or decrypted data, according to Mode, size is InputLen bytes.
pTag	Pointer to object that receives the tag calculated over data.
TagLen	Octet length of the authentication tag.
pInput	Data to be encrypted or decrypted.
InputLen	Octet length of the input data to be encrypted.
pAAD	Pointer to additional data authenticated by tag but not encrypted.
AADLen	Octet length of the additional data.
pIV	Pointer to initialization vector.
IVLen	Octet length of the initialization vector.
Encrypt	Nonzero for encryption, zero for decryption.
pAPI	Pointer to CIPHER API for ciphering.

Chapter 8

Storage device encryption

emCrypt implements the following storage device encryption algorithms:

- XTS-AES
- XTS-ARIA
- XTS-Camellia
- XTS-SEED
- XTS-Twofish

8.1 XTS-AES

8.1.1 Standards reference

XTS-AES is specified by the following document:

- NIST SP 800-38E — *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*

8.1.2 Type-safe API

Function	Description
<code>CRYPTO_XTS_AES_Encrypt()</code>	Encipher using XTS-AES.
<code>CRYPTO_XTS_AES_Decrypt()</code>	Decipher using XTS-AES.

8.1.2.1 CRYPTO_XTS_AES_Encrypt()

Description

Encipher using XTS-AES.

Prototype

```
void CRYPTO_XTS_AES_Encrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the encrypted data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the plaintext data.
InputLen	Number of bytes of data to be encrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.1.2.2 CRYPTO_XTS_AES_Decrypt()

Description

Decipher using XTS-AES.

Prototype

```
void CRYPTO_XTS_AES_Decrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the plaintext data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the encrypted data.
InputLen	Number of bytes of data to be decrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.2 XTS-ARIA

8.2.1 Type-safe API

Function	Description
CRYPTO_XTS_ARIA_Encrypt()	Encipher using XTS-ARIA.
CRYPTO_XTS_ARIA_Decrypt()	Decipher using XTS-ARIA.

8.2.1.1 CRYPTO_XTS_ARIA_Encrypt()

Description

Encipher using XTS-ARIA.

Prototype

```
void CRYPTO_XTS_ARIA_Encrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the encrypted data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the plaintext data.
InputLen	Number of bytes of data to be encrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.2.1.2 CRYPTO_XTS_ARIA_Decrypt()

Description

Decipher using XTS-ARIA.

Prototype

```
void CRYPTO_XTS_ARIA_Decrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the plaintext data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the encrypted data.
InputLen	Number of bytes of data to be decrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.3 XTS-Camellia

8.3.1 Type-safe API

Function	Description
<code>CRYPTO_XTS_CAMELLIA_Encrypt()</code>	Encipher using XTS-Camellia.
<code>CRYPTO_XTS_CAMELLIA_Decrypt()</code>	Decipher using XTS-Camellia.

8.3.1.1 CRYPTO_XTS_CAMELLIA_Encrypt()

Description

Encipher using XTS-Camellia.

Prototype

```
void CRYPTO_XTS_CAMELLIA_Encrypt(      U8      * pOutput,
                                         U64      UnitNumber,
                                         const U8      * pInput,
                                         unsigned   InputLen,
                                         const U8      * pKey1,
                                         const U8      * pKey2,
                                         unsigned   KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the encrypted data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the plaintext data.
InputLen	Number of bytes of data to be encrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.3.1.2 CRYPTO_XTS_CAMELLIA_Decrypt()

Description

Decipher using XTS-Camellia.

Prototype

```
void CRYPTO_XTS_CAMELLIA_Decrypt(      U8      * pOutput,
                                         U64      UnitNumber,
                                         const U8      * pInput,
                                         unsigned   InputLen,
                                         const U8      * pKey1,
                                         const U8      * pKey2,
                                         unsigned   KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the plaintext data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the encrypted data.
InputLen	Number of bytes of data to be decrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.4 XTS-SEED

8.4.1 Type-safe API

Function	Description
<code>CRYPTO_XTS_SEED_Encrypt()</code>	Encipher using XTS-SEED.
<code>CRYPTO_XTS_SEED_Decrypt()</code>	Decipher using XTS-SEED.

8.4.1.1 CRYPTO_XTS_SEED_Encrypt()

Description

Encipher using XTS-SEED.

Prototype

```
void CRYPTO_XTS_SEED_Encrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the encrypted data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the plaintext data.
InputLen	Number of bytes of data to be encrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.4.1.2 CRYPTO_XTS_SEED_Decrypt()

Description

Decipher using XTS-SEED.

Prototype

```
void CRYPTO_XTS_SEED_Decrypt(      U8      * pOutput,
                                    U64      UnitNumber,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKey1,
                                    const U8      * pKey2,
                                    unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the plaintext data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the encrypted data.
InputLen	Number of bytes of data to be decrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.5 XTS-Twofish

8.5.1 Type-safe API

Function	Description
<code>CRYPTO_XTS_TWOFISH_Encrypt()</code>	Encipher using XTS-Twofish.
<code>CRYPTO_XTS_TWOFISH_Decrypt()</code>	Decipher using XTS-Twofish.

8.5.1.1 CRYPTO_XTS_TWOFISH_Encrypt()

Description

Encipher using XTS-Twofish.

Prototype

```
void CRYPTO_XTS_TWOFISH_Encrypt(      U8      * pOutput,
                                         U64      UnitNumber,
                                         const U8      * pInput,
                                         unsigned   InputLen,
                                         const U8      * pKey1,
                                         const U8      * pKey2,
                                         unsigned   KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the encrypted data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the plaintext data.
InputLen	Number of bytes of data to be encrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

8.5.1.2 CRYPTO_XTS_TWOFISH_Decrypt()

Description

Decipher using XTS-Twofish.

Prototype

```
void CRYPTO_XTS_TWOFISH_Decrypt(      U8      * pOutput,
                                         U64      UnitNumber,
                                         const U8      * pInput,
                                         unsigned     InputLen,
                                         const U8      * pKey1,
                                         const U8      * pKey2,
                                         unsigned     KeyLen);
```

Parameters

Parameter	Description
pOutput	Pointer to buffer that receives the plaintext data.
UnitNumber	Data unit number for tweak.
pInput	Pointer to buffer that contains the encrypted data.
InputLen	Number of bytes of data to be decrypted in bytes; must be a multiple of 16.
pKey1	Pointer to key for data decryption.
pKey2	Pointer to key for tweak decryption.
KeyLen	Octet length of ciphering keys pKey1 and pKey2.

Chapter 9

Random bit generation

emCrypt implements the following deterministic random bit generators:

- Fortuna
- Hash-DRBG-SHA-1
- Hash-DRBG-SHA-224
- Hash-DRBG-SHA-256
- Hash-DRBG-SHA-384
- Hash-DRBG-SHA-512
- Hash-DRBG-SHA-512/224
- Hash-DRBG-SHA-512/256
- HMAC-DRBG-SHA-1
- HMAC-DRBG-SHA-224
- HMAC-DRBG-SHA-256
- HMAC-DRBG-SHA-384
- HMAC-DRBG-SHA-512
- HMAC-DRBG-SHA-512/224
- HMAC-DRBG-SHA-512/256
- CTR-DRBG-TDES
- CTR-DRBG-AES-128
- CTR-DRBG-AES-192
- CTR-DRBG-AES-256

9.1 Fortuna

9.1.1 Type-safe API

Function	Description
<code>CRYPTO_FORTUNA_Init()</code>	Initialize Fortuna context.
<code>CRYPTO_FORTUNA_Kill()</code>	Deinitialize Fortuna context.
<code>CRYPTO_FORTUNA_Status()</code>	Return Fortuna RNG status.
<code>CRYPTO_FORTUNA_Add()</code>	Add entropy.
<code>CRYPTO_FORTUNA_AddEx()</code>	Add entropy to pool.
<code>CRYPTO_FORTUNA_Reseed()</code>	Reseed Fortuna context.
<code>CRYPTO_FORTUNA_Get()</code>	Get pseudorandom data.

9.1.2 CRYPTO_FORTUNA_Add()

Description

Add entropy.

Prototype

```
void CRYPTO_FORTUNA_Add(      CRYPTO_FORTUNA_CONTEXT * pSelf,
                               unsigned                 Source,
                               const U8                * pData,
                               unsigned                 DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.
Source	Source index of random data.
pData	Pointer to octet string of random data.
DataLen	Octet length of octet string.

9.1.3 CRYPTO_FORTUNA_AddEx()

Description

Add entropy to pool.

Prototype

```
void CRYPTO_FORTUNA_AddEx( CRYPTO_FORTUNA_CONTEXT * pSelf,  
                            unsigned Source,  
                            unsigned Pool,  
                            const U8 * pData,  
                            unsigned DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.
Source	Source index of random data.
Pool	Pool to add to entropy to.
pData	Pointer to octet string of random data.
DataLen	Octet length of octet string.

9.1.4 CRYPTO_FORTUNA_Get()

Description

Get pseudorandom data.

Prototype

```
void CRYPTO_FORTUNA_Get(CRYPTO_FORTUNA_CONTEXT * pSelf,  
                         U8 * pData,  
                         unsigned DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.
pData	Pointer to object that receives the random data.
DataLen	Octet length of the object.

9.1.5 CRYPTO_FORTUNA_Init()

Description

Initialize Fortuna context.

Prototype

```
void CRYPTO_FORTUNA_Init(CRYPTO_FORTUNA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.

9.1.6 CRYPTO_FORTUNA_Kill()

Description

Deinitialize Fortuna context.

Prototype

```
void CRYPTO_FORTUNA_Kill(CRYPTO_FORTUNA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.

9.1.7 CRYPTO_FORTUNA_Reseed()

Description

Reseed Fortuna context.

Prototype

```
void CRYPTO_FORTUNA_Reseed(CRYPTO_FORTUNA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.

9.1.8 CRYPTO_FORTUNA_Status()

Description

Return Fortuna RNG status.

Prototype

```
int CRYPTO_FORTUNA_Status(CRYPTO_FORTUNA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Fortuna context.

Return value

- > 0 Fortuna is ready to deliver PRNG data.
- = 0 Fortuna needs reseeding.
- < 0 Fortuna does not have enough entropy.

9.1.9 Self-test API

The following table lists the Fortuna self-test API functions.

Function	Description
CRYPTO_FORTUNA_Voss_SelfTest()	Run Fortuna KATs defined by Voss.

9.1.9.1 CRYPTO_FORTUNA_Voss_SelfTest()

Description

Run Fortuna KATs defined by Voss.

Prototype

```
void CRYPTO_FORTUNA_Voss_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.2 Hash-DRBG-SHA-1

9.2.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.2.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA1_Init()</code>	Initialize a Hash-DRBG-SHA-1 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA1_Reseed()</code>	Reseed a HMAC-DRBG-SHA-1 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA1_Get()</code>	Get data from random bitstream.

9.2.2.1 CRYPTO_DRBG_HASH_SHA1_Init()

Description

Initialize a Hash-DRBG-SHA-1 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA1_Init(          CRYPTO_DRBG_HASH_SHA1_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pNonce,
                                         unsigned                     NonceLen,
                                         const U8                      * pPerso,
                                         unsigned                     PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.2.2.2 CRYPTO_DRBG_HASH_SHA1_Reseed()

Description

Reseed a HMAC-DRBG-SHA-1 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA1_Reseed(      CRYPTO_DRBG_HASH_SHA1_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pAdd,
                                         unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.2.2.3 CRYPTO_DRBG_HASH_SHA1_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA1_Get( CRYPTO_DRBG_HASH_SHA1_CONTEXT * pSelf,  
                                U8                      * pOutput,  
                                unsigned                 OutputLen,  
                                const U8                * pAdd,  
                                unsigned                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.2.3 Self-test API

The following table lists the Hash-DRBG-SHA-1 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA1_CAVS_SelfTest()	Run DRBG-SHA-1 KATs from CAVS.

9.2.3.1 CRYPTO_DRBG_HASH_SHA1_CAVS_SelfTest()

Description

Run DRBG-SHA-1 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA1_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.3 Hash-DRBG-SHA-224

9.3.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.3.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA224_Init()</code>	Initialize a Hash-DRBG-SHA-224 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA224_Reseed()</code>	Reseed a HMAC-DRBG-SHA-224 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA224_Get()</code>	Get data from random bitstream.

9.3.2.1 CRYPTO_DRBG_HASH_SHA224_Init()

Description

Initialize a Hash-DRBG-SHA-224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA224_Init(
    CRYPTO_DRBG_HASH_SHA224_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.3.2.2 CRYPTO_DRBG_HASH_SHA224_Reseed()

Description

Reseed a HMAC-DRBG-SHA-224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA224_Reseed
    (
        CRYPTO_DRBG_HASH_SHA224_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.3.2.3 CRYPTO_DRBG_HASH_SHA224_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA224_Get( CRYPTO_DRBG_HASH_SHA224_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.3.3 Self-test API

The following table lists the Hash-DRBG-SHA-224 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA224_CAVS_SelfTest	(Run DRBG-SHA-224 KATs from CAVS.

9.3.3.1 CRYPTO_DRBG_HASH_SHA224_CAVS_SelfTest()

Description

Run DRBG-SHA-224 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.4 Hash-DRBG-SHA-256

9.4.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.4.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA256_Init()</code>	Initialize a Hash-DRBG-SHA-256 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA256_Reseed()</code>	Reseed a HMAC-DRBG-SHA-256 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA256_Get()</code>	Get data from random bitstream.

9.4.2.1 CRYPTO_DRBG_HASH_SHA256_Init()

Description

Initialize a Hash-DRBG-SHA-256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA256_Init(
    CRYPTO_DRBG_HASH_SHA256_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.4.2.2 CRYPTO_DRBG_HASH_SHA256_Reseed()

Description

Reseed a HMAC-DRBG-SHA-256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA256_Reseed
    (
        CRYPTO_DRBG_HASH_SHA256_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.4.2.3 CRYPTO_DRBG_HASH_SHA256_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA256_Get( CRYPTO_DRBG_HASH_SHA256_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.4.3 Self-test API

The following table lists the Hash-DRBG-SHA-256 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA256_CAVS_SelfTest	(Run DRBG-SHA-256 KATs from CAVS.

9.4.3.1 CRYPTO_DRBG_HASH_SHA256_CAVS_SelfTest()

Description

Run DRBG-SHA-256 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.5 Hash-DRBG-SHA-384

9.5.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.5.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA384_Init()</code>	Initialize a Hash-DRBG-SHA-384 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA384_Reseed()</code>	Reseed a HMAC-DRBG-SHA-384 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA384_Get()</code>	Get data from random bitstream.

9.5.2.1 CRYPTO_DRBG_HASH_SHA384_Init()

Description

Initialize a Hash-DRBG-SHA-384 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA384_Init(
    CRYPTO_DRBG_HASH_SHA384_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.5.2.2 CRYPTO_DRBG_HASH_SHA384_Reseed()

Description

Reseed a HMAC-DRBG-SHA-384 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA384_Reseed
    (
        CRYPTO_DRBG_HASH_SHA384_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.5.2.3 CRYPTO_DRBG_HASH_SHA384_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA384_Get( CRYPTO_DRBG_HASH_SHA384_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.5.3 Self-test API

The following table lists the Hash-DRBG-SHA-384 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA384_CAVS_SelfTest	(Run DRBG-SHA-384 KATs from CAVS.

9.5.3.1 CRYPTO_DRBG_HASH_SHA384_CAVS_SelfTest()

Description

Run DRBG-SHA-384 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.6 Hash-DRBG-SHA-512

9.6.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.6.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA512_Init()</code>	Initialize a Hash-DRBG-SHA-512 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_Get()</code>	Get data from random bitstream.

9.6.2.1 CRYPTO_DRBG_HASH_SHA512_Init()

Description

Initialize a Hash-DRBG-SHA-512 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_Init
    (
        CRYPTO_DRBG_HASH_SHA512_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pNonce,
        unsigned                           NonceLen,
        const U8                         * pPerso,
        unsigned                           PersoLen);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to DRBG context.
<code>pEntropy</code>	Pointer to initial entropy input octet string.
<code>EntropyLen</code>	Octet length of the entropy input octet string.
<code>pNonce</code>	Pointer to nonce octet string.
<code>NonceLen</code>	Octet length of the nonce octet string.
<code>pPerso</code>	Pointer to personalization octet string.
<code>PersoLen</code>	Octet length of the personalization octet string.

9.6.2.2 CRYPTO_DRBG_HASH_SHA512_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_Reseed
    (
        CRYPTO_DRBG_HASH_SHA512_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.6.2.3 CRYPTO_DRBG_HASH_SHA512_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_Get(          CRYPTO_DRBG_HASH_SHA512_CONTEXT * pSelf,  
                                     U8                           * pOutput,  
                                     unsigned                      OutputLen,  
                                     const U8                      * pAdd,  
                                     unsigned                      AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.6.3 Self-test API

The following table lists the Hash-DRBG-SHA-512 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA512_CAVS_SelfTest	(Run DRBG-SHA-512 KATs from CAVS.

9.6.3.1 CRYPTO_DRBG_HASH_SHA512_CAVS_SelfTest()

Description

Run DRBG-SHA-512 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.7 Hash-DRBG-SHA-512/224

9.7.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.7.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA512_224_Init()</code>	Initialize a Hash-DRBG-SHA-512/224 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_224_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512/224 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_224_Get()</code>	Get data from random bitstream.

9.7.2.1 CRYPTO_DRBG_HASH_SHA512_224_Init()

Description

Initialize a Hash-DRBG-SHA-512/224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_224_Init
(
    CRYPTO_DRBG_HASH_SHA512_224_CONTEXT * pSelf,
    const U8                           * pEntropy,
    unsigned                           EntropyLen,
    const U8                           * pNonce,
    unsigned                           NonceLen,
    const U8                           * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.7.2.2 CRYPTO_DRBG_HASH_SHA512_224_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512/224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_224_Reseed
(          CRYPTO_DRBG_HASH_SHA512_224_CONTEXT * pSelf,
           const U8                                * pEntropy,
           unsigned                                 EntropyLen,
           const U8                                * pAdd,
           unsigned                                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.7.2.3 CRYPTO_DRBG_HASH_SHA512_224_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_224_Get
    (      CRYPTO_DRBG_HASH_SHA512_224_CONTEXT * pSelf,
          U8                                * pOutput,
          unsigned                           OutputLen,
          const U8                            * pAdd,
          unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.7.3 Self-test API

The following table lists the Hash-DRBG-SHA-512/224 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA512_224_CAVS_SelfTest	Run DRBG-SHA-512/224 KATs from CAVS.

9.7.3.1 CRYPTO_DRBG_HASH_SHA512_224_CAVS_SelfTest()

Description

Run DRBG-SHA-512/224 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.8 Hash-DRBG-SHA-512/256

9.8.1 Standards reference

Hash-DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.8.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HASH_SHA512_256_Init()</code>	Initialize a Hash-DRBG-SHA-512/256 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_256_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512/256 random bit generator.
<code>CRYPTO_DRBG_HASH_SHA512_256_Get()</code>	Get data from random bitstream.

9.8.2.1 CRYPTO_DRBG_HASH_SHA512_256_Init()

Description

Initialize a Hash-DRBG-SHA-512/256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_256_Init
(
    CRYPTO_DRBG_HASH_SHA512_256_CONTEXT * pSelf,
    const U8                           * pEntropy,
    unsigned                           EntropyLen,
    const U8                           * pNonce,
    unsigned                           NonceLen,
    const U8                           * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.8.2.2 CRYPTO_DRBG_HASH_SHA512_256_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512/256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_256_Reseed
(          CRYPTO_DRBG_HASH_SHA512_256_CONTEXT * pSelf,
           const U8                                * pEntropy,
           unsigned                                 EntropyLen,
           const U8                                * pAdd,
           unsigned                                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.8.2.3 CRYPTO_DRBG_HASH_SHA512_256_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_256_Get
    (      CRYPTO_DRBG_HASH_SHA512_256_CONTEXT * pSelf,
          U8                                * pOutput,
          unsigned                           OutputLen,
          const U8                            * pAdd,
          unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.8.3 Self-test API

The following table lists the Hash-DRBG-SHA-512/256 self-test API functions.

Function	Description
CRYPTO_DRBG_HASH_SHA512_256_CAVS_SelfTest	Run DRBG-SHA-512/256 KATs from CAVS.

9.8.3.1 CRYPTO_DRBG_HASH_SHA512_256_CAVS_SelfTest()

Description

Run DRBG-SHA-512/256 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HASH_SHA512_256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.9 HMAC-DRBG-SHA-1

9.9.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.9.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA1_Init()</code>	Initialize a HMAC-DRBG-SHA-1 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA1_Reseed()</code>	Reseed a HMAC-DRBG-SHA-1 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA1_Get()</code>	Get data from random bitstream.

9.9.2.1 CRYPTO_DRBG_HMAC_SHA1_Init()

Description

Initialize a HMAC-DRBG-SHA-1 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA1_Init(          CRYPTO_DRBG_HMAC_SHA1_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pNonce,
                                         unsigned                     NonceLen,
                                         const U8                      * pPerso,
                                         unsigned                     PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.9.2.2 CRYPTO_DRBG_HMAC_SHA1_Reseed()

Description

Reseed a HMAC-DRBG-SHA-1 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA1_Reseed(      CRYPTO_DRBG_HMAC_SHA1_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pAdd,
                                         unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.9.2.3 CRYPTO_DRBG_HMAC_SHA1_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA1_Get(          CRYPTO_DRBG_HMAC_SHA1_CONTEXT * pSelf,
                                         U8                           * pOutput,
                                         unsigned                      OutputLen,
                                         const U8                      * pAdd,
                                         unsigned                      AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.9.3 Self-test API

The following table lists the HMAC-DRBG-SHA-1 self-test API functions.

Function	Description
CRYPTO_DRBG_HMAC_SHA1_CAVS_SelfTest()	Run DRBG-HMAC-SHA-1 KATs from CAVS.

9.9.3.1 CRYPTO_DRBG_HMAC_SHA1_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-1 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA1_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.10 HMAC-DRBG-SHA-224

9.10.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.10.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA224_Init()</code>	Initialize a HMAC-DRBG-SHA-224 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA224_Reseed()</code>	Reseed a HMAC-DRBG-SHA-224 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA224_Get()</code>	Get data from random bitstream.

9.10.2.1 CRYPTO_DRBG_HMAC_SHA224_Init()

Description

Initialize a HMAC-DRBG-SHA-224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA224_Init(
    CRYPTO_DRBG_HMAC_SHA224_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.10.2.2 CRYPTO_DRBG_HMAC_SHA224_Reseed()

Description

Reseed a HMAC-DRBG-SHA-224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA224_Reseed
    (
        CRYPTO_DRBG_HMAC_SHA224_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.10.2.3 CRYPTO_DRBG_HMAC_SHA224_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA224_Get( CRYPTO_DRBG_HMAC_SHA224_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.10.3 Self-test API

The following table lists the HMAC-DRBG-SHA-224 self-test API functions.

Function	Description
CRYPTO_DRBG_HMAC_SHA224_CAVS_SelfTest()	Run DRBG-HMAC-SHA-224 KATs from CAVS.

9.10.3.1 CRYPTO_DRBG_HMAC_SHA224_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-224 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.11 HMAC-DRBG-SHA-256

9.11.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.11.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA256_Init()</code>	Initialize a HMAC-DRBG-SHA-256 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA256_Reseed()</code>	Reseed a HMAC-DRBG-SHA-256 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA256_Get()</code>	Get data from random bitstream.

9.11.2.1 CRYPTO_DRBG_HMAC_SHA256_Init()

Description

Initialize a HMAC-DRBG-SHA-256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA256_Init(
    CRYPTO_DRBG_HMAC_SHA256_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.11.2.2 CRYPTO_DRBG_HMAC_SHA256_Reseed()

Description

Reseed a HMAC-DRBG-SHA-256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA256_Reseed
(
    CRYPTO_DRBG_HMAC_SHA256_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pAdd,
    unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.11.2.3 CRYPTO_DRBG_HMAC_SHA256_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA256_Get( CRYPTO_DRBG_HMAC_SHA256_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.11.3 Self-test API

The following table lists the HMAC-DRBG-SHA-256 self-test API functions.

Function	Description
CRYPTO_DRBG_HMAC_SHA256_CAVS_SelfTest()	Run DRBG-HMAC-SHA-256 KATs from CAVS.

9.11.3.1 CRYPTO_DRBG_HMAC_SHA256_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-256 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.12 HMAC-DRBG-SHA-384

9.12.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.12.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA384_Init()</code>	Initialize a HMAC-DRBG-SHA-384 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA384_Reseed()</code>	Reseed a HMAC-DRBG-SHA-384 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA384_Get()</code>	Get data from random bitstream.

9.12.2.1 CRYPTO_DRBG_HMAC_SHA384_Init()

Description

Initialize a HMAC-DRBG-SHA-384 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA384_Init(
    CRYPTO_DRBG_HMAC_SHA384_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.12.2.2 CRYPTO_DRBG_HMAC_SHA384_Reseed()

Description

Reseed a HMAC-DRBG-SHA-384 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA384_Reseed
    (
        CRYPTO_DRBG_HMAC_SHA384_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.12.2.3 CRYPTO_DRBG_HMAC_SHA384_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA384_Get( CRYPTO_DRBG_HMAC_SHA384_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.12.3 Self-test API

The following table lists the HMAC-DRBG-SHA-384 self-test API functions.

Function	Description
CRYPTO_DRBG_HMAC_SHA384_CAVS_SelfTest()	Run DRBG-HMAC-SHA-384 KATs from CAVS.

9.12.3.1 CRYPTO_DRBG_HMAC_SHA384_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-384 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.13 HMAC-DRBG-SHA-512

9.13.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.13.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA512_Init()</code>	Initialize a HMAC-DRBG-SHA-512 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_Get()</code>	Get data from random bitstream.

9.13.2.1 CRYPTO_DRBG_HMAC_SHA512_Init()

Description

Initialize a HMAC-DRBG-SHA-512 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_Init(
    CRYPTO_DRBG_HMAC_SHA512_CONTEXT * pSelf,
    const U8                         * pEntropy,
    unsigned                           EntropyLen,
    const U8                         * pNonce,
    unsigned                           NonceLen,
    const U8                         * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.13.2.2 CRYPTO_DRBG_HMAC_SHA512_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_Reseed
    (
        CRYPTO_DRBG_HMAC_SHA512_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.13.2.3 CRYPTO_DRBG_HMAC_SHA512_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_Get( CRYPTO_DRBG_HMAC_SHA512_CONTEXT * pSelf,  
                                    U8                         * pOutput,  
                                    unsigned                     OutputLen,  
                                    const U8                      * pAdd,  
                                    unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.13.3 Self-test API

The following table lists the HMAC-DRBG-SHA-512 self-test API functions.

Function	Description
CRYPTO_DRBG_HMAC_SHA512_CAVS_SelfTest()	Run DRBG-HMAC-SHA-512 KATs from CAVS.

9.13.3.1 CRYPTO_DRBG_HMAC_SHA512_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-512 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.14 HMAC-DRBG-SHA-512/224

9.14.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.14.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA512_224_Init()</code>	Initialize a HMAC-DRBG-SHA-512/224 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_224_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512/224 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_224_Get()</code>	Get data from random bitstream.

9.14.2.1 CRYPTO_DRBG_HMAC_SHA512_224_Init()

Description

Initialize a HMAC-DRBG-SHA-512/224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_224_Init
(
    CRYPTO_DRBG_HMAC_SHA512_224_CONTEXT * pSelf,
    const U8                           * pEntropy,
    unsigned                           EntropyLen,
    const U8                           * pNonce,
    unsigned                           NonceLen,
    const U8                           * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.14.2.2 CRYPTO_DRBG_HMAC_SHA512_224_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512/224 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_224_Reseed
(          CRYPTO_DRBG_HMAC_SHA512_224_CONTEXT * pSelf,
    const U8                                * pEntropy,
    unsigned                                 EntropyLen,
    const U8                                * pAdd,
    unsigned                                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.14.2.3 CRYPTO_DRBG_HMAC_SHA512_224_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_224_Get
    (    CRYPTO_DRBG_HMAC_SHA512_224_CONTEXT * pSelf,
        U8                                * pOutput,
        unsigned                           OutputLen,
        const U8                            * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.14.3 Self-test API

The following table lists the HMAC-DRBG-SHA-512/224 self-test API functions.

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA512_224_CAVS_SelfTest()</code>	Run DRBG-HMAC-SHA-512/224 KATs from CAVS.

9.14.3.1 CRYPTO_DRBG_HMAC_SHA512_224_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-512/224 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.15 HMAC-DRBG-SHA-512/256

9.15.1 Standards reference

HMAC_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.15.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA512_256_Init()</code>	Initialize a HMAC-DRBG-SHA-512/256 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_256_Reseed()</code>	Reseed a HMAC-DRBG-SHA-512/256 random bit generator.
<code>CRYPTO_DRBG_HMAC_SHA512_256_Get()</code>	Get data from random bitstream.

9.15.2.1 CRYPTO_DRBG_HMAC_SHA512_256_Init()

Description

Initialize a HMAC-DRBG-SHA-512/256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_256_Init
(
    CRYPTO_DRBG_HMAC_SHA512_256_CONTEXT * pSelf,
    const U8                           * pEntropy,
    unsigned                           EntropyLen,
    const U8                           * pNonce,
    unsigned                           NonceLen,
    const U8                           * pPerso,
    unsigned                           PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.15.2.2 CRYPTO_DRBG_HMAC_SHA512_256_Reseed()

Description

Reseed a HMAC-DRBG-SHA-512/256 random bit generator.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_256_Reseed
(          CRYPTO_DRBG_HMAC_SHA512_256_CONTEXT * pSelf,
           const U8                                * pEntropy,
           unsigned                                 EntropyLen,
           const U8                                * pAdd,
           unsigned                                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.15.2.3 CRYPTO_DRBG_HMAC_SHA512_256_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_256_Get
    (    CRYPTO_DRBG_HMAC_SHA512_256_CONTEXT * pSelf,
        U8                                * pOutput,
        unsigned                           OutputLen,
        const U8                            * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.15.3 Self-test API

The following table lists the HMAC-DRBG-SHA-512/256 self-test API functions.

Function	Description
<code>CRYPTO_DRBG_HMAC_SHA512_256_CAVS_SelfTest()</code>	Run DRBG-HMAC-SHA-512/256 KATs from CAVS.

9.15.3.1 CRYPTO_DRBG_HMAC_SHA512_256_CAVS_SelfTest()

Description

Run DRBG-HMAC-SHA-512/256 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_HMAC_SHA512_256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.16 CTR-DRBG-TDES

9.16.1 Standards reference

CTR_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.16.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_CTR_TDES_Init()</code>	Initialize a CTR-DRBG-TDES random bit generator.
<code>CRYPTO_DRBG_CTR_TDES_Reseed()</code>	Reseed a CTR-DRBG-TDES random bit generator.
<code>CRYPTO_DRBG_CTR_TDES_Get()</code>	Get data from random bitstream.

9.16.2.1 CRYPTO_DRBG_CTR_TDES_Init()

Description

Initialize a CTR-DRBG-TDES random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_TDES_Init(      CRYPTO_DRBG_CTR_TDES_CONTEXT * pSelf,
                                         const U8                           * pEntropy,
                                         unsigned                         EntropyLen,
                                         const U8                           * pNonce,
                                         unsigned                         NonceLen,
                                         const U8                           * pPerso,
                                         unsigned                         PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.16.2.2 CRYPTO_DRBG_CTR_TDES_Reseed()

Description

Reseed a CTR-DRBG-TDES random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_TDES_Reseed(      CRYPTO_DRBG_CTR_TDES_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pAdd,
                                         unsigned                     AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.16.2.3 CRYPTO_DRBG_CTR_TDES_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_CTR_TDES_Get( CRYPTO_DRBG_CTR_TDES_CONTEXT * pSelf,  
                                U8 * pOutput,  
                                unsigned OutputLen,  
                                const U8 * pAdd,  
                                unsigned AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.16.3 Self-test API

The following table lists the DRBG-CTR-TDES self-test API functions.

Function	Description
CRYPTO_DRBG_CTR_TDES_CAVS_SelfTest()	Run DRBG-CTR-TDES KATs from CAVS.

9.16.3.1 CRYPTO_DRBG_CTR_TDES_CAVS_SelfTest()

Description

Run DRBG-CTR-TDES KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_CTR_TDES_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.17 CTR-DRBG-AES-128

9.17.1 Standards reference

CTR_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.17.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_CTR_AES128_Init()</code>	Initialize a CTR-DRBG-AES-128 random bit generator.
<code>CRYPTO_DRBG_CTR_AES128_Reseed()</code>	Reseed a CTR-DRBG-AES-128 random bit generator.
<code>CRYPTO_DRBG_CTR_AES128_Get()</code>	Get data from random bitstream.

9.17.2.1 CRYPTO_DRBG_CTR_AES128_Init()

Description

Initialize a CTR-DRBG-AES-128 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES128_Init(      CRYPTO_DRBG_CTR_AES128_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pNonce,
                                         unsigned                     NonceLen,
                                         const U8                      * pPerso,
                                         unsigned                     PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.17.2.2 CRYPTO_DRBG_CTR_AES128_Reseed()

Description

Reseed a CTR-DRBG-AES-128 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES128_Reseed
    (
        CRYPTO_DRBG_CTR_AES128_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.17.2.3 CRYPTO_DRBG_CTR_AES128_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_CTR_AES128_Get( CRYPTO_DRBG_CTR_AES128_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  unsigned                 OutputLen,  
                                  const U8                * pAdd,  
                                  unsigned                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.17.3 Self-test API

The following table lists the DRBG-CTR-AES-128 self-test API functions.

Function	Description
CRYPTO_DRBG_CTR_AES128_CAVS_SelfTest()	Run DRBG-CTR-AES-128 KATs from CAVS.

9.17.3.1 CRYPTO_DRBG_CTR_AES128_CAVS_SelfTest()

Description

Run DRBG-CTR-AES-128 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_CTR_AES128_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.18 CTR-DRBG-AES-192

9.18.1 Standards reference

CTR_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.18.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_CTR_AES192_Init()</code>	Initialize a CTR-DRBG-AES-192 random bit generator.
<code>CRYPTO_DRBG_CTR_AES192_Reseed()</code>	Reseed a CTR-DRBG-AES-192 random bit generator.
<code>CRYPTO_DRBG_CTR_AES192_Get()</code>	Get data from random bitstream.

9.18.2.1 CRYPTO_DRBG_CTR_AES192_Init()

Description

Initialize a CTR-DRBG-AES-192 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES192_Init(          CRYPTO_DRBG_CTR_AES192_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pNonce,
                                         unsigned                     NonceLen,
                                         const U8                      * pPerso,
                                         unsigned                     PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.18.2.2 CRYPTO_DRBG_CTR_AES192_Reseed()

Description

Reseed a CTR-DRBG-AES-192 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES192_Reseed
    (
        CRYPTO_DRBG_CTR_AES192_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.18.2.3 CRYPTO_DRBG_CTR_AES192_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_CTR_AES192_Get( CRYPTO_DRBG_CTR_AES192_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  unsigned                 OutputLen,  
                                  const U8                * pAdd,  
                                  unsigned                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.18.3 Self-test API

The following table lists the DRBG-CTR-AES-192 self-test API functions.

Function	Description
CRYPTO_DRBG_CTR_AES192_CAVS_SelfTest()	Run DRBG-CTR-AES-192 KATs from CAVS.

9.18.3.1 CRYPTO_DRBG_CTR_AES192_CAVS_SelfTest()

Description

Run DRBG-CTR-AES-192 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_CTR_AES192_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

9.19 CTR-DRBG-AES-256

9.19.1 Standards reference

CTR_DRBG is specified by the following document:

- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

9.19.2 Type-safe API

Function	Description
<code>CRYPTO_DRBG_CTR_AES256_Init()</code>	Initialize a CTR-DRBG-AES-256 random bit generator.
<code>CRYPTO_DRBG_CTR_AES256_Reseed()</code>	Reseed a CTR-DRBG-AES-256 random bit generator.
<code>CRYPTO_DRBG_CTR_AES256_Get()</code>	Get data from random bitstream.

9.19.2.1 CRYPTO_DRBG_CTR_AES256_Init()

Description

Initialize a CTR-DRBG-AES-256 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES256_Init(      CRYPTO_DRBG_CTR_AES256_CONTEXT * pSelf,
                                         const U8                      * pEntropy,
                                         unsigned                     EntropyLen,
                                         const U8                      * pNonce,
                                         unsigned                     NonceLen,
                                         const U8                      * pPerso,
                                         unsigned                     PersoLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input octet string.
EntropyLen	Octet length of the entropy input octet string.
pNonce	Pointer to nonce octet string.
NonceLen	Octet length of the nonce octet string.
pPerso	Pointer to personalization octet string.
PersoLen	Octet length of the personalization octet string.

9.19.2.2 CRYPTO_DRBG_CTR_AES256_Reseed()

Description

Reseed a CTR-DRBG-AES-265 random bit generator.

Prototype

```
void CRYPTO_DRBG_CTR_AES256_Reseed
    (
        CRYPTO_DRBG_CTR_AES256_CONTEXT * pSelf,
        const U8                         * pEntropy,
        unsigned                           EntropyLen,
        const U8                         * pAdd,
        unsigned                           AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pEntropy	Pointer to initial entropy input string.
EntropyLen	Octet length of the entropy input octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.19.2.3 CRYPTO_DRBG_CTR_AES256_Get()

Description

Get data from random bitstream.

Prototype

```
void CRYPTO_DRBG_CTR_AES256_Get( CRYPTO_DRBG_CTR_AES256_CONTEXT * pSelf,  
                                  U8                      * pOutput,  
                                  unsigned                 OutputLen,  
                                  const U8                * pAdd,  
                                  unsigned                 AddLen);
```

Parameters

Parameter	Description
pSelf	Pointer to DRBG context.
pOutput	Pointer to object that receives the random data.
OutputLen	Octet length of the random data octet string.
pAdd	Pointer to additional input octet string.
AddLen	Octet length of the additional input octet string.

9.19.3 Self-test API

The following table lists the DRBG-CTR-AES-256 self-test API functions.

Function	Description
CRYPTO_DRBG_CTR_AES256_CAVS_SelfTest()	Run DRBG-CTR-AES-256 KATs from CAVS.

9.19.3.1 CRYPTO_DRBG_CTR_AES256_CAVS_SelfTest()

Description

Run DRBG-CTR-AES-256 KATs from CAVS.

Prototype

```
void CRYPTO_DRBG_CTR_AES256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

Chapter 10

Key derivation

emCrypt implements the following key derivation algorithms:

- KDF1
- KDF2
- X9.63 KDF
- HKDF
- PBKDF2

Although NIST has recommendation for other key derivation algorithms, such as the IKE, TLS, and SSH key derivation functions, these key derivation functions are recommended by NIST only in their specific application domain. Therefore emCrypt does not provide these functions and leaves it to individual products to implement any application-specific key derivation function.

10.1 KDF1

10.1.1 Standards reference

KDF1 is specified by the following documents:

- ISO/IEC 18033-2 — *Encryption algorithms — Part 2: Asymmetric ciphers*
- IEEE Std 1363-2000 — *IEEE Standard Specifications for Public-Key Cryptography*
- IETF RFC 2437 — *PKCS #1: RSA Cryptography Specifications Version 2.2*

10.1.2 Alternative naming

The function KDF1 is also known as MGF1 in P1363, PKCS #1, and some other standards. MGF1 and KDF1 are identical.

10.1.3 Type-safe API

Function	Description
Plain key derivation	
<code>CRYPTO_KDF1_SHA1_Calc()</code>	Derive key using KDF1-SHA-1.
<code>CRYPTO_KDF1_SHA224_Calc()</code>	Derive key using KDF1-SHA-224.
<code>CRYPTO_KDF1_SHA256_Calc()</code>	Derive key using KDF1-SHA-256.
<code>CRYPTO_KDF1_SHA384_Calc()</code>	Derive key using KDF1-SHA-384.
<code>CRYPTO_KDF1_SHA512_Calc()</code>	Derive key using KDF1-SHA-512.
<code>CRYPTO_KDF1_SHA512_224_Calc()</code>	Derive key using KDF1-SHA-512/224.
<code>CRYPTO_KDF1_SHA512_256_Calc()</code>	Derive key using KDF1-SHA-512/256.
<code>CRYPTO_KDF1_SHA3_224_Calc()</code>	Derive key using KDF1-SHA-224.
<code>CRYPTO_KDF1_SHA3_256_Calc()</code>	Derive key using KDF1-SHA-256.
<code>CRYPTO_KDF1_SHA3_384_Calc()</code>	Derive key using KDF1-SHA-384.
<code>CRYPTO_KDF1_SHA3_512_Calc()</code>	Derive key using KDF1-SHA-512.
<code>CRYPTO_KDF1_SM3_Calc()</code>	Derive key using KDF1-SM3.
<code>CRYPTO_KDF1_BLAKE2B_Calc()</code>	Derive key using KDF1-BLAKE2b.
<code>CRYPTO_KDF1_BLAKE2S_Calc()</code>	Derive key using KDF1-BLAKE2s.
Derive key and combine	
<code>CRYPTO_KDF1_SHA1_CalcEx()</code>	Derive key using KDF1-SHA-1, combine output.
<code>CRYPTO_KDF1_SHA224_CalcEx()</code>	Derive key using KDF1-SHA-224, combine output.
<code>CRYPTO_KDF1_SHA256_CalcEx()</code>	Derive key using KDF1-SHA-256, combine output.
<code>CRYPTO_KDF1_SHA384_CalcEx()</code>	Derive key using KDF1-SHA-384, combine output.
<code>CRYPTO_KDF1_SHA512_CalcEx()</code>	Derive key using KDF1-SHA-512, combine output.
<code>CRYPTO_KDF1_SHA512_224_CalcEx()</code>	Derive key using KDF1-SHA-512/224, combine output.
<code>CRYPTO_KDF1_SHA512_256_CalcEx()</code>	Derive key using KDF1-SHA-512/256, combine output.
<code>CRYPTO_KDF1_SHA3_224_CalcEx()</code>	Derive key using KDF1-SHA-224, combine output.

Function	Description
<code>CRYPTO_KDF1_SHA3_256_CalcEx()</code>	Derive key using KDF1-SHA-256, combine output.
<code>CRYPTO_KDF1_SHA3_384_CalcEx()</code>	Derive key using KDF1-SHA-384, combine output.
<code>CRYPTO_KDF1_SHA3_512_CalcEx()</code>	Derive key using KDF1-SHA-512, combine output.
<code>CRYPTO_KDF1_SM3_CalcEx()</code>	Derive key using KDF1-SM3, combine output.
<code>CRYPTO_KDF1_BLAKE2B_CalcEx()</code>	Derive key using KDF1-BLAKE2b, combine output.
<code>CRYPTO_KDF1_BLAKE2S_CalcEx()</code>	Derive key using KDF1-BLAKE2s, combine output.

10.1.3.1 CRYPTO_KDF1_SHA1_Calc()

Description

Derive key using KDF1-SHA-1.

Prototype

```
void CRYPTO_KDF1_SHA1_Calc(const U8      * pSeed,
                           unsigned   SeedLen,
                           U8      * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.2 CRYPTO_KDF1_SHA1_CalcEx()

Description

Derive key using KDF1-SHA-1, combine output.

Prototype

```
void CRYPTO_KDF1_SHA1_CalcEx(const U8 * pSeed,
                               unsigned SeedLen,
                               U8 * pOutput,
                               unsigned OutputLen,
                               CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.3 CRYPTO_KDF1_SHA224_Calc()

Description

Derive key using KDF1-SHA-224.

Prototype

```
void CRYPTO_KDF1_SHA224_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.4 CRYPTO_KDF1_SHA224_CalcEx()

Description

Derive key using KDF1-SHA-224, combine output.

Prototype

```
void CRYPTO_KDF1_SHA224_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
          unsigned SeedLen,  
          unsigned OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.5 CRYPTO_KDF1_SHA256_Calc()

Description

Derive key using KDF1-SHA-256.

Prototype

```
void CRYPTO_KDF1_SHA256_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.6 CRYPTO_KDF1_SHA256_CalcEx()

Description

Derive key using KDF1-SHA-256, combine output.

Prototype

```
void CRYPTO_KDF1_SHA256_CalcEx(const U8 * pSeed,
                                 unsigned SeedLen,
                                 U8 * pOutput,
                                 unsigned OutputLen,
                                 CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.7 CRYPTO_KDF1_SHA384_Calc()

Description

Derive key using KDF1-SHA-384.

Prototype

```
void CRYPTO_KDF1_SHA384_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.8 CRYPTO_KDF1_SHA384_CalcEx()

Description

Derive key using KDF1-SHA-384, combine output.

Prototype

```
void CRYPTO_KDF1_SHA384_CalcEx(const U8 * pSeed,
                                 unsigned SeedLen,
                                 U8 * pOutput,
                                 unsigned OutputLen,
                                 CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.9 CRYPTO_KDF1_SHA512_Calc()

Description

Derive key using KDF1-SHA-512.

Prototype

```
void CRYPTO_KDF1_SHA512_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.10 CRYPTO_KDF1_SHA512_CalcEx()

Description

Derive key using KDF1-SHA-512, combine output.

Prototype

```
void CRYPTO_KDF1_SHA512_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
          unsigned SeedLen,           * pOutput,  
          unsigned OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.11 CRYPTO_KDF1_SHA512_224_Calc()

Description

Derive key using KDF1-SHA-512/224.

Prototype

```
void CRYPTO_KDF1_SHA512_224_Calc(const U8      * pSeed,
                                    unsigned   SeedLen,
                                    U8      * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.12 CRYPTO_KDF1_SHA512_224_CalcEx()

Description

Derive key using KDF1-SHA-512/224, combine output.

Prototype

```
void CRYPTO_KDF1_SHA512_224_CalcEx(const U8
                                     unsigned          * pSeed,
                                     U8                 SeedLen,
                                     unsigned          * pOutput,
                                     CRYPTO_LOGIC_OP   OutputLen,
                                     Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.13 CRYPTO_KDF1_SHA512_256_Calc()

Description

Derive key using KDF1-SHA-512/256.

Prototype

```
void CRYPTO_KDF1_SHA512_256_Calc(const U8      * pSeed,
                                    unsigned   SeedLen,
                                    U8      * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.14 CRYPTO_KDF1_SHA512_256_CalcEx()

Description

Derive key using KDF1-SHA-512/256, combine output.

Prototype

```
void CRYPTO_KDF1_SHA512_256_CalcEx(const U8
                                     unsigned          * pSeed,
                                     U8                 SeedLen,
                                     unsigned          * pOutput,
                                     CRYPTO_LOGIC_OP   OutputLen,
                                     Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.15 CRYPTO_KDF1_SHA3_224_Calc()

Description

Derive key using KDF1-SHA-224.

Prototype

```
void CRYPTO_KDF1_SHA3_224_Calc(const U8      * pSeed,
                                 unsigned     SeedLen,
                                 U8          * pOutput,
                                 unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.16 CRYPTO_KDF1_SHA3_224_CalcEx()

Description

Derive key using KDF1-SHA-224, combine output.

Prototype

```
void CRYPTO_KDF1_SHA3_224_CalcEx(const U8 * pSeed,
                                    unsigned SeedLen,
                                    U8 * pOutput,
                                    unsigned OutputLen,
                                    CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.17 CRYPTO_KDF1_SHA3_256_Calc()

Description

Derive key using KDF1-SHA-256.

Prototype

```
void CRYPTO_KDF1_SHA3_256_Calc(const U8      * pSeed,
                                 unsigned     SeedLen,
                                 U8          * pOutput,
                                 unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.18 CRYPTO_KDF1_SHA3_256_CalcEx()

Description

Derive key using KDF1-SHA-256, combine output.

Prototype

```
void CRYPTO_KDF1_SHA3_256_CalcEx(const U8 * pSeed,
                                    unsigned SeedLen,
                                    U8 * pOutput,
                                    unsigned OutputLen,
                                    CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.19 CRYPTO_KDF1_SHA3_384_Calc()

Description

Derive key using KDF1-SHA-384.

Prototype

```
void CRYPTO_KDF1_SHA3_384_Calc(const U8      * pSeed,
                                 unsigned     SeedLen,
                                 U8          * pOutput,
                                 unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.20 CRYPTO_KDF1_SHA3_384_CalcEx()

Description

Derive key using KDF1-SHA-384, combine output.

Prototype

```
void CRYPTO_KDF1_SHA3_384_CalcEx(const U8 * pSeed,
                                    unsigned SeedLen,
                                    U8 * pOutput,
                                    unsigned OutputLen,
                                    CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.21 CRYPTO_KDF1_SHA3_512_Calc()

Description

Derive key using KDF1-SHA-512.

Prototype

```
void CRYPTO_KDF1_SHA3_512_Calc(const U8      * pSeed,
                                 unsigned     SeedLen,
                                 U8          * pOutput,
                                 unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.22 CRYPTO_KDF1_SHA3_512_CalcEx()

Description

Derive key using KDF1-SHA-512, combine output.

Prototype

```
void CRYPTO_KDF1_SHA3_512_CalcEx(const U8 * pSeed,
                                    unsigned SeedLen,
                                    U8 * pOutput,
                                    unsigned OutputLen,
                                    CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.23 CRYPTO_KDF1_SM3_Calc()

Description

Derive key using KDF1-SM3.

Prototype

```
void CRYPTO_KDF1_SM3_Calc(const U8      * pSeed,
                           unsigned   SeedLen,
                           U8      * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.24 CRYPTO_KDF1_SM3_CalcEx()

Description

Derive key using KDF1-SM3, combine output.

Prototype

```
void CRYPTO_KDF1_SM3_CalcEx(const U8 * pSeed,  
                           unsigned SeedLen,  
                           U8 * pOutput,  
                           unsigned OutputLen,  
                           CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.25 CRYPTO_KDF1_BLAKE2B_Calc()

Description

Derive key using KDF1-BLAKE2b.

Prototype

```
void CRYPTO_KDF1_BLAKE2B_Calc(const U8      * pSeed,
                                unsigned     SeedLen,
                                U8          * pOutput,
                                unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.26 CRYPTO_KDF1_BLAKE2B_CalcEx()

Description

Derive key using KDF1-BLAKE2b, combine output.

Prototype

```
void CRYPTO_KDF1_BLAKE2B_CalcEx(const U8 * pSeed,  
                                  U8 SeedLen,  
                                  U8 * pOutput,  
                                  U8 OutputLen,  
                                  CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.1.3.27 CRYPTO_KDF1_BLAKE2S_Calc()

Description

Derive key using KDF1-BLAKE2s.

Prototype

```
void CRYPTO_KDF1_BLAKE2S_Calc(const U8      * pSeed,
                                unsigned     SeedLen,
                                U8          * pOutput,
                                unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.1.3.28 CRYPTO_KDF1_BLAKE2S_CalcEx()

Description

Derive key using KDF1-BLAKE2s, combine output.

Prototype

```
void CRYPTO_KDF1_BLAKE2S_CalcEx(const U8 * pSeed,  
                                  U8 SeedLen,  
                                  U8 * pOutput,  
                                  U8 OutputLen,  
                                  CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2 KDF2

10.2.1 Standards reference

KDF2 is specified by the following documents:

- ISO/IEC 18033-2 — *Encryption algorithms — Part 2: Asymmetric ciphers*
- IEEE Std 1363-2000 — *IEEE Standard Specifications for Public-Key Cryptography*

10.2.2 Type-safe API

Function	Description
<code>CRYPTO_KDF2_SHA1_Calc()</code>	Derive key using KDF2-SHA-1.
<code>CRYPTO_KDF2_SHA224_Calc()</code>	Derive key using KDF2-SHA-224.
<code>CRYPTO_KDF2_SHA256_Calc()</code>	Derive key using KDF2-SHA-256.
<code>CRYPTO_KDF2_SHA384_Calc()</code>	Derive key using KDF2-SHA-384.
<code>CRYPTO_KDF2_SHA512_Calc()</code>	Derive key using KDF2-SHA-512.
<code>CRYPTO_KDF2_SHA512_224_Calc()</code>	Derive key using KDF2-SHA-512/224.
<code>CRYPTO_KDF2_SHA512_256_Calc()</code>	Derive key using KDF2-SHA-512/256.
<code>CRYPTO_KDF2_SHA3_224_Calc()</code>	Derive key using KDF2-SHA3-224.
<code>CRYPTO_KDF2_SHA3_256_Calc()</code>	Derive key using KDF2-SHA3-256.
<code>CRYPTO_KDF2_SHA3_384_Calc()</code>	Derive key using KDF2-SHA3-384.
<code>CRYPTO_KDF2_SHA3_512_Calc()</code>	Derive key using KDF2-SHA3-512.
<code>CRYPTO_KDF2_SM3_Calc()</code>	Derive key using KDF2-SM3.
<code>CRYPTO_KDF2_BLAKE2B_Calc()</code>	Derive key using KDF2-BLAKE2b.
<code>CRYPTO_KDF2_BLAKE2S_Calc()</code>	Derive key using KDF2-BLAKE2s.
Derive key and combine	
<code>CRYPTO_KDF2_SHA1_CalcEx()</code>	Derive key using KDF2-SHA-1, combine output.
<code>CRYPTO_KDF2_SHA224_CalcEx()</code>	Derive key using KDF2-SHA-224, combine output.
<code>CRYPTO_KDF2_SHA256_CalcEx()</code>	Derive key using KDF2-SHA-256, combine output.
<code>CRYPTO_KDF2_SHA384_CalcEx()</code>	Derive key using KDF2-SHA-384, combine output.
<code>CRYPTO_KDF2_SHA512_CalcEx()</code>	Derive key using KDF2-SHA-512, combine output.
<code>CRYPTO_KDF2_SHA512_224_CalcEx()</code>	Derive key using KDF2-SHA-512/224, combine output.
<code>CRYPTO_KDF2_SHA512_256_CalcEx()</code>	Derive key using KDF2-SHA-512/256, combine output.
<code>CRYPTO_KDF2_SHA3_224_CalcEx()</code>	Derive key using KDF2-SHA3-224, combine output.
<code>CRYPTO_KDF2_SHA3_256_CalcEx()</code>	Derive key using KDF2-SHA3-256, combine output.
<code>CRYPTO_KDF2_SHA3_384_CalcEx()</code>	Derive key using KDF2-SHA3-384, combine output.
<code>CRYPTO_KDF2_SHA3_512_CalcEx()</code>	Derive key using KDF2-SHA3-512, combine output.

Function	Description
CRYPTO_KDF2_SM3_CalcEx()	Derive key using KDF2-SM3, combine output.
CRYPTO_KDF2_BLAKE2B_CalcEx()	Derive key using KDF2-BLAKE2b, combine output.
CRYPTO_KDF2_BLAKE2S_CalcEx()	Derive key using KDF2-BLAKE2s, combine output.

10.2.2.1 CRYPTO_KDF2_SHA1_Calc()

Description

Derive key using KDF2-SHA-1.

Prototype

```
void CRYPTO_KDF2_SHA1_Calc(const U8      * pSeed,
                           unsigned   SeedLen,
                           U8      * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.2 CRYPTO_KDF2_SHA1_CalcEx()

Description

Derive key using KDF2-SHA-1, combine output.

Prototype

```
void CRYPTO_KDF2_SHA1_CalcEx(const U8 * pSeed,
                               unsigned SeedLen,
                               U8 * pOutput,
                               unsigned OutputLen,
                               CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.3 CRYPTO_KDF2_SHA224_Calc()

Description

Derive key using KDF2-SHA-224.

Prototype

```
void CRYPTO_KDF2_SHA224_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.4 CRYPTO_KDF2_SHA224_CalcEx()

Description

Derive key using KDF2-SHA-224, combine output.

Prototype

```
void CRYPTO_KDF2_SHA224_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
          unsigned SeedLen,           * pOutput,  
          unsigned OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.5 CRYPTO_KDF2_SHA256_Calc()

Description

Derive key using KDF2-SHA-256.

Prototype

```
void CRYPTO_KDF2_SHA256_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.6 CRYPTO_KDF2_SHA256_CalcEx()

Description

Derive key using KDF2-SHA-256, combine output.

Prototype

```
void CRYPTO_KDF2_SHA256_CalcEx(const U8          * pSeed,
                                unsigned           SeedLen,
                                U8                * pOutput,
                                unsigned           OutputLen,
                                CRYPTO_LOGIC_OP   Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.7 CRYPTO_KDF2_SHA384_Calc()

Description

Derive key using KDF2-SHA-384.

Prototype

```
void CRYPTO_KDF2_SHA384_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.8 CRYPTO_KDF2_SHA384_CalcEx()

Description

Derive key using KDF2-SHA-384, combine output.

Prototype

```
void CRYPTO_KDF2_SHA384_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
          unsigned SeedLen,  
          unsigned OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.9 CRYPTO_KDF2_SHA512_Calc()

Description

Derive key using KDF2-SHA-512.

Prototype

```
void CRYPTO_KDF2_SHA512_Calc(const U8      * pSeed,
                               unsigned     SeedLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.10 CRYPTO_KDF2_SHA512_CalcEx()

Description

Derive key using KDF2-SHA-512, combine output.

Prototype

```
void CRYPTO_KDF2_SHA512_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
          unsigned SeedLen,           * pOutput,  
          unsigned OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.11 CRYPTO_KDF2_SHA512_224_Calc()

Description

Derive key using KDF2-SHA-512/224.

Prototype

```
void CRYPTO_KDF2_SHA512_224_Calc(const U8      * pSeed,
                                    unsigned   SeedLen,
                                    U8      * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.12 CRYPTO_KDF2_SHA512_224_CalcEx()

Description

Derive key using KDF2-SHA-512/224, combine output.

Prototype

```
void CRYPTO_KDF2_SHA512_224_CalcEx(const U8
                                     unsigned          * pSeed,
                                     U8                 SeedLen,
                                     unsigned          * pOutput,
                                     CRYPTO_LOGIC_OP   OutputLen,
                                     Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.13 CRYPTO_KDF2_SHA512_256_Calc()

Description

Derive key using KDF2-SHA-512/256.

Prototype

```
void CRYPTO_KDF2_SHA512_256_Calc(const U8      * pSeed,
                                    unsigned   SeedLen,
                                    U8      * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.14 CRYPTO_KDF2_SHA512_256_CalcEx()

Description

Derive key using KDF2-SHA-512/256, combine output.

Prototype

```
void CRYPTO_KDF2_SHA512_256_CalcEx(const U8
                                      unsigned          * pSeed,
                                      SeedLen,           * pOutput,
                                      U8                 OutputLen,
                                      unsigned          Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.15 CRYPTO_KDF2_SHA3_224_Calc()

Description

Derive key using KDF2-SHA3-224.

Prototype

```
void CRYPTO_KDF2_SHA3_224_Calc(const U8      * pSeed,
                                  unsigned     SeedLen,
                                  U8          * pOutput,
                                  unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.16 CRYPTO_KDF2_SHA3_224_CalcEx()

Description

Derive key using KDF2-SHA3-224, combine output.

Prototype

```
void CRYPTO_KDF2_SHA3_224_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
U8  
unsigned  
unsigned  
SeedLen,  
OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.17 CRYPTO_KDF2_SHA3_256_Calc()

Description

Derive key using KDF2-SHA3-256.

Prototype

```
void CRYPTO_KDF2_SHA3_256_Calc(const U8      * pSeed,
                                  unsigned   SeedLen,
                                  U8      * pOutput,
                                  unsigned  OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.18 CRYPTO_KDF2_SHA3_256_CalcEx()

Description

Derive key using KDF2-SHA3-256, combine output.

Prototype

```
void CRYPTO_KDF2_SHA3_256_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
U8  
unsigned  
unsigned  
SeedLen,  
OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.19 CRYPTO_KDF2_SHA3_384_Calc()

Description

Derive key using KDF2-SHA3-384.

Prototype

```
void CRYPTO_KDF2_SHA3_384_Calc(const U8      * pSeed,
                                  unsigned   SeedLen,
                                  U8      * pOutput,
                                  unsigned  OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.20 CRYPTO_KDF2_SHA3_384_CalcEx()

Description

Derive key using KDF2-SHA3-384, combine output.

Prototype

```
void CRYPTO_KDF2_SHA3_384_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
U8  
unsigned  
unsigned  
SeedLen,  
OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.21 CRYPTO_KDF2_SHA3_512_Calc()

Description

Derive key using KDF2-SHA3-512.

Prototype

```
void CRYPTO_KDF2_SHA3_512_Calc(const U8      * pSeed,
                                  unsigned     SeedLen,
                                  U8          * pOutput,
                                  unsigned    OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.22 CRYPTO_KDF2_SHA3_512_CalcEx()

Description

Derive key using KDF2-SHA3-512, combine output.

Prototype

```
void CRYPTO_KDF2_SHA3_512_CalcEx(const U8 * pSeed, U8 * pOutput, CRYPTO_LOGIC_OP Operation);  
U8  
unsigned  
unsigned  
SeedLen,  
OutputLen,
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.23 CRYPTO_KDF2_SM3_Calc()

Description

Derive key using KDF2-SM3.

Prototype

```
void CRYPTO_KDF2_SM3_Calc(const U8      * pSeed,
                           unsigned   SeedLen,
                           U8      * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.24 CRYPTO_KDF2_SM3_CalcEx()

Description

Derive key using KDF2-SM3, combine output.

Prototype

```
void CRYPTO_KDF2_SM3_CalcEx(const U8 * pSeed,  
                           unsigned SeedLen,  
                           U8 * pOutput,  
                           unsigned OutputLen,  
                           CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.25 CRYPTO_KDF2_BLAKE2B_Calc()

Description

Derive key using KDF2-BLAKE2b.

Prototype

```
void CRYPTO_KDF2_BLAKE2B_Calc(const U8      * pSeed,
                                unsigned     SeedLen,
                                U8          * pOutput,
                                unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.26 CRYPTO_KDF2_BLAKE2B_CalcEx()

Description

Derive key using KDF2-BLAKE2b, combine output.

Prototype

```
void CRYPTO_KDF2_BLAKE2B_CalcEx(const U8 * pSeed,  
                                  U8 SeedLen,  
                                  U8 * pOutput,  
                                  U8 OutputLen,  
                                  CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.2.2.27 CRYPTO_KDF2_BLAKE2S_Calc()

Description

Derive key using KDF2-BLAKE2s.

Prototype

```
void CRYPTO_KDF2_BLAKE2S_Calc(const U8      * pSeed,
                                unsigned     SeedLen,
                                U8          * pOutput,
                                unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.2.2.28 CRYPTO_KDF2_BLAKE2S_CalcEx()

Description

Derive key using KDF2-BLAKE2s, combine output.

Prototype

```
void CRYPTO_KDF2_BLAKE2S_CalcEx(const U8 * pSeed,  
                                  U8 SeedLen,  
                                  U8 * pOutput,  
                                  U8 OutputLen,  
                                  CRYPTO_LOGIC_OP Operation);
```

Parameters

Parameter	Description
pSeed	Pointer to seed for mask generation.
SeedLen	Octet length of the seed.
pOutput	Pointer to buffer to receive computed mask.
OutputLen	Octet length of the buffer.
Operation	Logical operation combining derived key with output.

Additional information

The output of the key derivation process is combined with the receiving object using the logical operation Operation.

10.3 X9.63 KDF

10.3.1 Standards reference

The X9.63 KDF is specified by the following document:

- ANS X9.63 — *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*

10.3.2 Type-safe API

Function	Description
Key derivation	
CRYPTO_X9v63_KDF_SHA1_Calc()	Derive key using X9.63 KDF-SHA-1.
CRYPTO_X9v63_KDF_SHA224_Calc()	Derive key using X9.63 KDF-SHA-224.
CRYPTO_X9v63_KDF_SHA256_Calc()	Derive key using X9.63 KDF-SHA-256.
CRYPTO_X9v63_KDF_SHA384_Calc()	Derive key using X9.63 KDF-SHA-384.
CRYPTO_X9v63_KDF_SHA512_Calc()	Derive key using X9.63 KDF-SHA-512.
CRYPTO_X9v63_KDF_SHA512_224_Calc()	Derive key using X9.63 KDF-SHA-512/224.
CRYPTO_X9v63_KDF_SHA512_256_Calc()	Derive key using X9.63 KDF-SHA-512/256.
CRYPTO_X9v63_KDF_SM3_Calc()	Derive key using X9.63 KDF-SM3.
CRYPTO_X9v63_KDF_BLAKE2B_Calc()	Derive key using X9.63 KDF-BLAKE2b.
CRYPTO_X9v63_KDF_BLAKE2S_Calc()	Derive key using X9.63 KDF-BLAKE2s.
Key derivation with shared data	
CRYPTO_X9v63_KDF_SHA1_CalcEx()	Derive key using X9.63 KDF-SHA-1, with shared data.
CRYPTO_X9v63_KDF_SHA224_CalcEx()	Derive key using X9.63 KDF-SHA-224, with shared data.
CRYPTO_X9v63_KDF_SHA256_CalcEx()	Derive key using X9.63 KDF-SHA-256, with shared data.
CRYPTO_X9v63_KDF_SHA384_CalcEx()	Derive key using X9.63 KDF-SHA-384, with shared data.
CRYPTO_X9v63_KDF_SHA512_CalcEx()	Derive key using X9.63 KDF-SHA-512, with shared data.
CRYPTO_X9v63_KDF_SHA512_224_CalcEx()	Derive key using X9.63 KDF-SHA-512/224, with shared data.
CRYPTO_X9v63_KDF_SHA512_256_CalcEx()	Derive key using X9.63 KDF-SHA-512/256, with shared data.
CRYPTO_X9v63_KDF_SM3_CalcEx()	Derive key using X9.63 KDF-SM3, with shared data.
CRYPTO_X9v63_KDF_BLAKE2B_CalcEx()	Derive key using X9.63 KDF-BLAKE2b, with shared data.
CRYPTO_X9v63_KDF_BLAKE2S_CalcEx()	Derive key using X9.63 KDF-BLAKE2s, with shared data.

10.3.2.1 CRYPTO_X9v63_KDF_SHA1_Calc()

Description

Derive key using X9.63 KDF-SHA-1.

Prototype

```
void CRYPTO_X9v63_KDF_SHA1_Calc(const U8      * pSeed,
                                  unsigned     SeedLen,
                                  U8      * pOutput,
                                  unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.2 CRYPTO_X9v63_KDF_SHA1_CalcEx()

Description

Derive key using X9.63 KDF-SHA-1, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA1_CalcEx(const U8      * pSeed,
                                    unsigned     SeedLen,
                                    const U8      * pShared,
                                    unsigned     SharedLen,
                                    U8          * pOutput,
                                    unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.3 CRYPTO_X9v63_KDF_SHA224_Calc()

Description

Derive key using X9.63 KDF-SHA-224.

Prototype

```
void CRYPTO_X9v63_KDF_SHA224_Calc(const U8      * pSeed,
                                     unsigned   SeedLen,
                                     U8      * pOutput,
                                     unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.4 CRYPTO_X9v63_KDF_SHA224_CalcEx()

Description

Derive key using X9.63 KDF-SHA-224, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA224_CalcEx(const U8      * pSeed,
                                       unsigned     SeedLen,
                                       const U8      * pShared,
                                       unsigned     SharedLen,
                                       U8          * pOutput,
                                       unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.5 CRYPTO_X9v63_KDF_SHA256_Calc()

Description

Derive key using X9.63 KDF-SHA-256.

Prototype

```
void CRYPTO_X9v63_KDF_SHA256_Calc(const U8      * pSeed,
                                     unsigned   SeedLen,
                                     U8      * pOutput,
                                     unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.6 CRYPTO_X9v63_KDF_SHA256_CalcEx()

Description

Derive key using X9.63 KDF-SHA-256, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA256_CalcEx(const U8      * pSeed,
                                       unsigned     SeedLen,
                                       const U8      * pShared,
                                       unsigned     SharedLen,
                                       U8          * pOutput,
                                       unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.7 CRYPTO_X9v63_KDF_SHA384_Calc()

Description

Derive key using X9.63 KDF-SHA-384.

Prototype

```
void CRYPTO_X9v63_KDF_SHA384_Calc(const U8      * pSeed,
                                    unsigned   SeedLen,
                                    U8      * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.8 CRYPTO_X9v63_KDF_SHA384_CalcEx()

Description

Derive key using X9.63 KDF-SHA-384, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA384_CalcEx(const U8      * pSeed,
                                       unsigned   SeedLen,
                                       const U8      * pShared,
                                       unsigned   SharedLen,
                                       U8        * pOutput,
                                       unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.9 CRYPTO_X9v63_KDF_SHA512_Calc()

Description

Derive key using X9.63 KDF-SHA-512.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_Calc(const U8      * pSeed,
                                     unsigned   SeedLen,
                                     U8      * pOutput,
                                     unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.10 CRYPTO_X9v63_KDF_SHA512_CalcEx()

Description

Derive key using X9.63 KDF-SHA-512, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_CalcEx(const U8      * pSeed,
                                       unsigned     SeedLen,
                                       const U8      * pShared,
                                       unsigned     SharedLen,
                                       U8          * pOutput,
                                       unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.11 CRYPTO_X9v63_KDF_SHA512_224_Calc()

Description

Derive key using X9.63 KDF-SHA-512/224.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_224_Calc(const U8      * pSeed,
                                         unsigned   SeedLen,
                                         U8      * pOutput,
                                         unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.12 CRYPTO_X9v63_KDF_SHA512_224_CalcEx()

Description

Derive key using X9.63 KDF-SHA-512/224, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_224_CalcEx(const U8      * pSeed,
                                             unsigned   SeedLen,
                                             const U8      * pShared,
                                             unsigned   SharedLen,
                                             U8      * pOutput,
                                             unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.13 CRYPTO_X9v63_KDF_SHA512_256_Calc()

Description

Derive key using X9.63 KDF-SHA-512/256.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_256_Calc(const U8      * pSeed,
                                         unsigned   SeedLen,
                                         U8      * pOutput,
                                         unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.14 CRYPTO_X9v63_KDF_SHA512_256_CalcEx()

Description

Derive key using X9.63 KDF-SHA-512/256, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SHA512_256_CalcEx(const U8      * pSeed,
                                             unsigned   SeedLen,
                                             const U8      * pShared,
                                             unsigned   SharedLen,
                                             U8      * pOutput,
                                             unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.15 CRYPTO_X9v63_KDF_SM3_Calc()

Description

Derive key using X9.63 KDF-SM3.

Prototype

```
void CRYPTO_X9v63_KDF_SM3_Calc(const U8      * pSeed,
                                  unsigned   SeedLen,
                                  U8      * pOutput,
                                  unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.16 CRYPTO_X9v63_KDF_SM3_CalcEx()

Description

Derive key using X9.63 KDF-SM3, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_SM3_CalcEx(const U8      * pSeed,
                                    unsigned     SeedLen,
                                    const U8      * pShared,
                                    unsigned     SharedLen,
                                    U8          * pOutput,
                                    unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.17 CRYPTO_X9v63_KDF_BLAKE2B_Calc()

Description

Derive key using X9.63 KDF-BLAKE2b.

Prototype

```
void CRYPTO_X9v63_KDF_BLAKE2B_Calc(const U8      * pSeed,
                                      unsigned   SeedLen,
                                      U8        * pOutput,
                                      unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.18 CRYPTO_X9v63_KDF_BLAKE2B_CalcEx()

Description

Derive key using X9.63 KDF-BLAKE2b, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_BLAKE2B_CalcEx(const U8      * pSeed,
                                         unsigned     SeedLen,
                                         const U8      * pShared,
                                         unsigned     SharedLen,
                                         U8          * pOutput,
                                         unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.19 CRYPTO_X9v63_KDF_BLAKE2S_Calc()

Description

Derive key using X9.63 KDF-BLAKE2s.

Prototype

```
void CRYPTO_X9v63_KDF_BLAKE2S_Calc(const U8      * pSeed,
                                      unsigned   SeedLen,
                                      U8        * pOutput,
                                      unsigned   OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.3.2.20 CRYPTO_X9v63_KDF_BLAKE2S_CalcEx()

Description

Derive key using X9.63 KDF-BLAKE2s, with shared data.

Prototype

```
void CRYPTO_X9v63_KDF_BLAKE2S_CalcEx(const U8      * pSeed,
                                         unsigned     SeedLen,
                                         const U8      * pShared,
                                         unsigned     SharedLen,
                                         U8          * pOutput,
                                         unsigned     OutputLen);
```

Parameters

Parameter	Description
pSeed	Pointer to seed octet string for key derivation.
SeedLen	Octet length of the seed octet string.
pShared	Pointer to shared octet string for key derivation.
SharedLen	Octet length of the shared octet string.
pOutput	Pointer to object that receives the derived key.
OutputLen	Octet length of the derived key.

10.4 HKDF

10.4.1 Type-safe API

Function	Description
Calculate derived key	
<code>CRYPTO_HKDF_MD5_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_RIPEMD160_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA1_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA224_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA256_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA384_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_224_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_256_Calc()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SM3_Calc()</code>	Compute pseudorandom key from keying material.
Extract operation	
<code>CRYPTO_HKDF_MD5_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_RIPEMD160_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA1_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA224_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA256_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA384_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_224_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SHA512_256_Extract()</code>	Compute pseudorandom key from keying material.
<code>CRYPTO_HKDF_SM3_Extract()</code>	Compute pseudorandom key from keying material.
Expand operation	
<code>CRYPTO_HKDF_MD5_Expand()</code>	Generate keying material from pseudorandom key.

Function	Description
<code>CRYPTO_HKDF_RIPEMD160_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA1_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA224_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA256_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA384_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA512_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA512_224_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SHA512_256_Expand()</code>	Generate keying material from pseudorandom key.
<code>CRYPTO_HKDF_SM3_Expand()</code>	Generate keying material from pseudorandom key.

10.4.1.1 CRYPTO_HKDF_MD5_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_MD5_Calc(const U8      * pInput,
                           unsigned   InputLen,
                           const U8      * pSalt,
                           unsigned   SaltLen,
                           const U8      * pInfo,
                           unsigned   InfoLen,
                           U8        * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.2 CRYPTO_HKDF_MD5_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_MD5_Expand(const U8      * pPRK,
                           unsigned        PRKLen,
                           const U8      * pInfo,
                           unsigned        InfoLen,
                           U8      * pOutput,
                           unsigned        OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.3 CRYPTO_HKDF_MD5_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_MD5_Extract(const U8      * pInput,
                             unsigned     InputLen,
                             const U8      * pSalt,
                             unsigned     SaltLen,
                             U8          * pPRK,
                             unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.4 CRYPTO_HKDF_RIPEMD160_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_RIPEMD160_Calc(const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8    * pSalt,
                                  unsigned     SaltLen,
                                  const U8    * pInfo,
                                  unsigned     InfoLen,
                                  U8          * pOutput,
                                  unsigned     OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.5 CRYPTO_HKDF_RIPEMD160_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_RIPEMD160_Expand(const U8      * pPRK,
                                    unsigned    PRKLen,
                                    const U8      * pInfo,
                                    unsigned    InfoLen,
                                    U8      * pOutput,
                                    unsigned    OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.6 CRYPTO_HKDF_RIPEMD160_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_RIPEMD160_Extract(const U8      * pInput,
                                     unsigned     InputLen,
                                     const U8      * pSalt,
                                     unsigned     SaltLen,
                                     U8          * pPRK,
                                     unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.7 CRYPTO_HKDF_SHA1_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA1_Calc(const U8      * pInput,
                           unsigned   InputLen,
                           const U8      * pSalt,
                           unsigned   SaltLen,
                           const U8      * pInfo,
                           unsigned   InfoLen,
                           U8        * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.8 CRYPTO_HKDF_SHA1_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA1_Expand(const U8      * pPRK,
                             unsigned     PRKLen,
                             const U8      * pInfo,
                             unsigned     InfoLen,
                             U8          * pOutput,
                             unsigned     OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.9 CRYPTO_HKDF_SHA1_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA1_Extract(const U8      * pInput,
                               unsigned     InputLen,
                               const U8      * pSalt,
                               unsigned     SaltLen,
                               U8          * pPRK,
                               unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.10 CRYPTO_HKDF_SHA224_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA224_Calc(const U8      * pInput,
                               unsigned     InputLen,
                               const U8      * pSalt,
                               unsigned     SaltLen,
                               const U8      * pInfo,
                               unsigned     InfoLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.11 CRYPTO_HKDF_SHA224_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA224_Expand(const U8          * pPRK,
                                unsigned        PRKLen,
                                const U8          * pInfo,
                                unsigned        InfoLen,
                                U8              * pOutput,
                                unsigned        OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.12 CRYPTO_HKDF_SHA224_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA224_Extract(const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8      * pSalt,
                                  unsigned     SaltLen,
                                  U8          * pPRK,
                                  unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.13 CRYPTO_HKDF_SHA256_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA256_Calc(const U8      * pInput,
                               unsigned     InputLen,
                               const U8      * pSalt,
                               unsigned     SaltLen,
                               const U8      * pInfo,
                               unsigned     InfoLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.14 CRYPTO_HKDF_SHA256_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA256_Expand(const U8          * pPRK,
                                unsigned        PRKLen,
                                const U8          * pInfo,
                                unsigned        InfoLen,
                                U8              * pOutput,
                                unsigned        OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.15 CRYPTO_HKDF_SHA256_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA256_Extract(const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8      * pSalt,
                                  unsigned     SaltLen,
                                  U8          * pPRK,
                                  unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.16 CRYPTO_HKDF_SHA384_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA384_Calc(const U8      * pInput,
                               unsigned     InputLen,
                               const U8      * pSalt,
                               unsigned     SaltLen,
                               const U8      * pInfo,
                               unsigned     InfoLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.17 CRYPTO_HKDF_SHA384_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA384_Expand(const U8          * pPRK,
                                unsigned        PRKLen,
                                const U8          * pInfo,
                                unsigned        InfoLen,
                                U8              * pOutput,
                                unsigned        OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.18 CRYPTO_HKDF_SHA384_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA384_Extract(const U8      * pInput,
                                 unsigned     InputLen,
                                 const U8      * pSalt,
                                 unsigned     SaltLen,
                                 U8          * pPRK,
                                 unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.19 CRYPTO_HKDF_SHA512_224_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_224_Calc(const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pSalt,
                                    unsigned     SaltLen,
                                    const U8      * pInfo,
                                    unsigned     InfoLen,
                                    U8          * pOutput,
                                    unsigned    OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.20 CRYPTO_HKDF_SHA512_224_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA512_224_Expand(const U8      * pPRK,
                                      unsigned    PRKLen,
                                      const U8      * pInfo,
                                      unsigned    InfoLen,
                                      U8      * pOutput,
                                      unsigned   OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.21 CRYPTO_HKDF_SHA512_224_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_224_Extract(const U8      * pInput,
                                      unsigned     InputLen,
                                      const U8      * pSalt,
                                      unsigned     SaltLen,
                                      U8          * pPRK,
                                      unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.22 CRYPTO_HKDF_SHA512_256_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_256_Calc(const U8      * pInput,
                                    unsigned   InputLen,
                                    const U8      * pSalt,
                                    unsigned   SaltLen,
                                    const U8      * pInfo,
                                    unsigned   InfoLen,
                                    U8        * pOutput,
                                    unsigned   OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.23 CRYPTO_HKDF_SHA512_256_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA512_256_Expand(const U8      * pPRK,
                                      unsigned    PRKLen,
                                      const U8      * pInfo,
                                      unsigned    InfoLen,
                                      U8      * pOutput,
                                      unsigned   OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.24 CRYPTO_HKDF_SHA512_256_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_256_Extract(const U8      * pInput,
                                      unsigned     InputLen,
                                      const U8      * pSalt,
                                      unsigned     SaltLen,
                                      U8          * pPRK,
                                      unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.25 CRYPTO_HKDF_SHA512_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_Calc(const U8      * pInput,
                               unsigned     InputLen,
                               const U8      * pSalt,
                               unsigned     SaltLen,
                               const U8      * pInfo,
                               unsigned     InfoLen,
                               U8          * pOutput,
                               unsigned    OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.26 CRYPTO_HKDF_SHA512_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SHA512_Expand(const U8          * pPRK,
                                unsigned           PRKLen,
                                const U8          * pInfo,
                                unsigned           InfoLen,
                                U8                * pOutput,
                                unsigned           OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.27 CRYPTO_HKDF_SHA512_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SHA512_Extract(const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8      * pSalt,
                                  unsigned     SaltLen,
                                  U8          * pPRK,
                                  unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.1.28 CRYPTO_HKDF_SM3_Calc()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SM3_Calc(const U8      * pInput,
                           unsigned   InputLen,
                           const U8      * pSalt,
                           unsigned   SaltLen,
                           const U8      * pInfo,
                           unsigned   InfoLen,
                           U8        * pOutput,
                           unsigned   OutputLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of salt.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.29 CRYPTO_HKDF_SM3_Expand()

Description

Generate keying material from pseudorandom key.

Prototype

```
void CRYPTO_HKDF_SM3_Expand(const U8      * pPRK,
                           unsigned     PRKLen,
                           const U8      * pInfo,
                           unsigned     InfoLen,
                           U8          * pOutput,
                           unsigned     OutputLen);
```

Parameters

Parameter	Description
pPRK	Pointer to pseudorandom key (usually the output of the extract step).
PRKLen	Octet length of the pseudorandom key.
pInfo	Pointer to context string.
InfoLen	Octet length of context string.
pOutput	Pointer to object that receives the output keying material.
OutputLen	Octet length of output keying material object.

10.4.1.30 CRYPTO_HKDF_SM3_Extract()

Description

Compute pseudorandom key from keying material.

Prototype

```
void CRYPTO_HKDF_SM3_Extract(const U8      * pInput,
                             unsigned     InputLen,
                             const U8      * pSalt,
                             unsigned     SaltLen,
                             U8          * pPRK,
                             unsigned     PRKLen);
```

Parameters

Parameter	Description
pInput	Pointer to input keying material.
InputLen	Octet length of keying material.
pSalt	Pointer to salt to use when hashing to avoid dictionary attacks.
SaltLen	Octet length of the salt.
pPRK	Pointer to object that receives the pseudorandom key.
PRKLen	Octet length of the pseudorandom key.

10.4.2 Self-test API

The following table lists the AESKW self-test API functions.

Function	Description
CRYPTO_HKDF_SelfTest()	Run all HKDF test vectors.

10.4.2.1 CRYPTO_HKDF_SelfTest()

Description

Run all HKDF test vectors.

Prototype

```
void CRYPTO_HKDF_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

10.5 PBKDF2

10.5.1 Type-safe API

Function	Description
<code>CRYPTO_PBKDF2_HMAC_SHA1_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA224_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA256_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA384_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA512_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA512_224_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SHA512_256_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.
<code>CRYPTO_PBKDF2_HMAC_SM3_Calc()</code>	Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

10.5.1.1 CRYPTO_PBKDF2_HMAC_SHA1_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA1_Calc(const U8      * pPassword,
                                    unsigned    PasswordLen,
                                    const U8      * pSalt,
                                    unsigned    SaltLen,
                                    unsigned    IterationCount,
                                    U8         * pOutput,
                                    unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.2 CRYPTO_PBKDF2_HMAC_SHA224_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA224_Calc(const U8      * pPassword,
                                       unsigned    PasswordLen,
                                       const U8    * pSalt,
                                       unsigned    SaltLen,
                                       unsigned    IterationCount,
                                       U8         * pOutput,
                                       unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.3 CRYPTO_PBKDF2_HMAC_SHA256_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA256_Calc(const U8      * pPassword,
                                      unsigned    PasswordLen,
                                      const U8    * pSalt,
                                      unsigned    SaltLen,
                                      unsigned    IterationCount,
                                      U8         * pOutput,
                                      unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.4 CRYPTO_PBKDF2_HMAC_SHA384_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA384_Calc(const U8      * pPassword,
                                      unsigned    PasswordLen,
                                      const U8    * pSalt,
                                      unsigned    SaltLen,
                                      unsigned    IterationCount,
                                      U8         * pOutput,
                                      unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.5 CRYPTO_PBKDF2_HMAC_SHA512_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA512_Calc(const U8      * pPassword,
                                       unsigned    PasswordLen,
                                       const U8    * pSalt,
                                       unsigned    SaltLen,
                                       unsigned    IterationCount,
                                       U8         * pOutput,
                                       unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.6 CRYPTO_PBKDF2_HMAC_SHA512_224_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA512_224_Calc(const U8      * pPassword,
                                         unsigned    PasswordLen,
                                         const U8      * pSalt,
                                         unsigned    SaltLen,
                                         unsigned    IterationCount,
                                         U8        * pOutput,
                                         unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.7 CRYPTO_PBKDF2_HMAC_SHA512_256_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SHA512_256_Calc(const U8      * pPassword,
                                         unsigned    PasswordLen,
                                         const U8      * pSalt,
                                         unsigned    SaltLen,
                                         unsigned    IterationCount,
                                         U8        * pOutput,
                                         unsigned    OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.1.8 CRYPTO_PBKDF2_HMAC_SM3_Calc()

Description

Generate a master key in Output[] derived from Password and Salt and the iteration count C using SHA-1 as the hash function.

Prototype

```
void CRYPTO_PBKDF2_HMAC_SM3_Calc(const U8          * pPassword,
                                  unsigned        PasswordLen,
                                  const U8        * pSalt,
                                  unsigned        SaltLen,
                                  unsigned        IterationCount,
                                  U8            * pOutput,
                                  unsigned        OutputLen);
```

Parameters

Parameter	Description
pPassword	Pointer to password octet string.
PasswordLen	Octet length of the password octet string.
pSalt	Pointer to salt octet string (avoiding dictionary attacks).
SaltLen	Octet length of the salt octet string.
IterationCount	Number of hashing iterations to perform.
pOutput	Pointer to object that receives the hashed password.
OutputLen	Octet length of the object that receives the hashed password.

10.5.2 Self-test API

The following table lists the PBKDF2 self-test API functions.

Function	Description
CRYPTO_PBKDF2_SelfTest()	Run all PBKDF2 test vectors.

10.5.2.1 CRYPTO_PBKDF2_SelfTest()

Description

Run all PBKDF2 test vectors.

Prototype

```
void CRYPTO_PBKDF2_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to selftest API.

Chapter 11

Extendable-output functions

emCrypt implements the following extendable-output functions:

- SHAKE128 and SHAKE256
- cSHAKE

In addition, the Keccak building block for SHA-3 and SHAKE is implemented.

11.1 SHAKE128

11.1.1 Type-safe API

Function	Description
Message functions	
CRYPTO_SHAKE128_Calc()	Calculate SHAKE128 output.
Incremental functions	
CRYPTO_SHAKE128_Init()	Initialize SHAKE128 context.
CRYPTO_SHAKE128_Add()	Add data to SHAKE128.
CRYPTO_SHAKE128_Final()	Add data to SHAKE128.
CRYPTO_SHAKE128_Kill()	Destroy SHAKE128 context.

11.1.1.1 CRYPTO_SHAKE128_Add()

Description

Add data to SHAKE128.

Prototype

```
void CRYPTO_SHAKE128_Add(      CRYPTO_SHAKE_CONTEXT * pSelf,
                               const U8                  * pInput,
                               unsigned                 InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

11.1.1.2 CRYPTO_SHAKE128_Calc()

Description

Calculate SHAKE128 output.

Prototype

```
void CRYPTO_SHAKE128_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8    * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

11.1.1.3 CRYPTO_SHAKE128_Final()

Description

Add data to SHAKE128.

Prototype

```
void CRYPTO_SHAKE128_Final(CRYPTO_SHAKE_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string.

11.1.1.4 CRYPTO_SHAKE128_Init()

Description

Initialize SHAKE128 context.

Prototype

```
void CRYPTO_SHAKE128_Init(CRYPTO_SHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.

11.1.1.5 CRYPTO_SHAKE128_Kill()

Description

Destroy SHAKE128 context.

Prototype

```
void CRYPTO_SHAKE128_Kill(CRYPTO_SHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.

11.1.2 Self-test API

The following table lists the SHAKE self-test API functions.

Function	Description
CRYPTO_SHAKE128_CAVS_SelfTest()	Run CAVS SHAKE128 self-test.

11.1.2.1 CRYPTO_SHAKE128_CAVS_SelfTest()

Description

Run CAVS SHAKE128 self-test.

Prototype

```
void CRYPTO_SHAKE128_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

11.2 SHAKE256

11.2.1 Type-safe API

Function	Description
Message functions	
CRYPTO_SHAKE256_Calc()	Calculate SHAKE256 output.
Incremental functions	
CRYPTO_SHAKE256_Init()	Initialize SHAKE256 context.
CRYPTO_SHAKE256_Add()	Add data to SHAKE256.
CRYPTO_SHAKE256_Final()	Add data to SHAKE256.
CRYPTO_SHAKE256_Kill()	Destroy SHAKE256 context.

11.2.1.1 CRYPTO_SHAKE256_Add()

Description

Add data to SHAKE256.

Prototype

```
void CRYPTO_SHAKE256_Add(      CRYPTO_SHAKE_CONTEXT * pSelf,
                               const U8             * pInput,
                               unsigned            InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

11.2.1.2 CRYPTO_SHAKE256_Calc()

Description

Calculate SHAKE256 output.

Prototype

```
void CRYPTO_SHAKE256_Calc(      U8      * pOutput,
                                unsigned   OutputLen,
                                const U8    * pInput,
                                unsigned   InputLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

11.2.1.3 CRYPTO_SHAKE256_Final()

Description

Add data to SHAKE256.

Prototype

```
void CRYPTO_SHAKE256_Final(CRYPTO_SHAKE_CONTEXT * pSelf,  
                           U8                      * pOutput,  
                           unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string.

11.2.1.4 CRYPTO_SHAKE256_Init()

Description

Initialize SHAKE256 context.

Prototype

```
void CRYPTO_SHAKE256_Init(CRYPTO_SHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.

11.2.1.5 CRYPTO_SHAKE256_Kill()

Description

Destroy SHAKE256 context.

Prototype

```
void CRYPTO_SHAKE256_Kill(CRYPTO_SHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SHAKE context.

11.2.2 Self-test API

The following table lists the SHAKE self-test API functions.

Function	Description
CRYPTO_SHAKE256_CAVS_SelfTest()	Run CAVS SHAKE256 self-test.

11.2.2.1 CRYPTO_SHAKE256_CAVS_SelfTest()

Description

Run CAVS SHAKE256 self-test.

Prototype

```
void CRYPTO_SHAKE256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

11.3 cSHAKE

11.3.1 Type-safe API

Function	Description
Message functions	
CRYPTO_CSHAKE128_Calc()	Calculate cSHAKE128 output.
CRYPTO_CSHAKE256_Calc()	Calculate cSHAKE256 output.
Incremental functions	
CRYPTO_CSHAKE_Init()	Initialize cSHAKE.
CRYPTO_CSHAKE_Add()	Add data (absorb).
CRYPTO_CSHAKE_Get()	Get data (squeeze).
CRYPTO_CSHAKE_Kill()	Clear cSHAKE context.
Building blocks	
CRYPTO_CSHAKE_LeftEncode()	Encode integer, left formatting.
CRYPTO_CSHAKE_RightEncode()	Encode integer, right formatting.
CRYPTO_CSHAKE_EncodeStr()	Encode octet string.
CRYPTO_CSHAKE_BlockPad()	Add zeros to block boundary.

11.3.1.1 CRYPTO_CSHAKE128_Calc()

Description

Calculate cSHAKE128 output.

Prototype

```
void CRYPTO_CSHAKE128_Calc(      U8      * pOutput,
                                  unsigned   OutputLen,
const U8      * pInput,
                                  unsigned   InputLen,
const U8      * pCust,
                                  unsigned   CustLen,
const U8      * pFunc,
                                  unsigned   FuncLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string (L/8).
pInput	Pointer to input octet string (X).
InputLen	Octet length of the input octet string.
pCust	Pointer to customization string (S).
CustLen	Octet length of the customization string.
pFunc	Pointer to NIST-allocated function name (N).
FuncLen	Octet length of the NIST-allocated function string.

11.3.1.2 CRYPTO_CSHAKE256_Calc()

Description

Calculate cSHAKE256 output.

Prototype

```
void CRYPTO_CSHAKE256_Calc(      U8      * pOutput,
                                  unsigned   OutputLen,
const U8      * pInput,
                                  unsigned   InputLen,
const U8      * pCust,
                                  unsigned   CustLen,
const U8      * pFunc,
                                  unsigned   FuncLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string (L/8).
pInput	Pointer to input octet string (X).
InputLen	Octet length of the input octet string.
pCust	Pointer to customization string (S).
CustLen	Octet length of the customization string.
pFunc	Pointer to NIST-allocated function name (N).
FuncLen	Octet length of the NIST-allocated function string.

11.3.1.3 CRYPTO_CSHAKE_Init()

Description

Initialize cSHAKE.

Prototype

```
void CRYPTO_CSHAKE_Init(      CRYPTO_CSHAKE_CONTEXT * pSelf,
                               const U8           * pCust,
                               unsigned           CustLen,
                               const U8           * pFunc,
                               unsigned           FuncLen,
                               unsigned           Security);
```

Parameters

Parameter	Description
pSelf	Pointer to cSHAKE context.
pCust	Pointer to customization string (S).
CustLen	Octet length of the customization string.
pFunc	Pointer to NIST-allocated function name (N).
FuncLen	Octet length of the NIST-allocated function string.
Security	Security strength in bits.

11.3.1.4 CRYPTO_CSHAKE_Add()

Description

Add data (absorb).

Prototype

```
void CRYPTO_CSHAKE_Add(          CRYPTO_CSHAKE_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cSHAKE context.
pInput	Pointer to input octet string.
InputLen	Octet length of the input octet string.

11.3.1.5 CRYPTO_CSHAKE_LeftEncode()

Description

Encode integer, left formatting.

Prototype

```
void CRYPTO_CSHAKE_LeftEncode(CRYPTO_CSHAKE_CONTEXT * pSelf,  
                               U32 N);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
N	Integer to encode.

11.3.1.6 CRYPTO_CSHAKE_RightEncode()

Description

Encode integer, right formatting.

Prototype

```
void CRYPTO_CSHAKE_RightEncode(CRYPTO_CSHAKE_CONTEXT * pSelf,  
                                U32 N);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
N	Integer to encode.

11.3.1.7 CRYPTO_CSHAKE_EncodeStr()

Description

Encode octet string.

Prototype

```
void CRYPTO_CSHAKE_EncodeStr(          CRYPTO_CSHAKE_CONTEXT * pSelf,
                                    const U8           * pStr,
                                    unsigned           StrLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
pStr	Pointer to octet string to encode.
StrLen	Octet length of the string to encode.

11.3.1.8 CRYPTO_CSHAKE_BlockPad()

Description

Add zeros to block boundary.

Prototype

```
void CRYPTO_CSHAKE_BlockPad(CRYPTO_CSHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cSHAKE context.

11.3.1.9 CRYPTO_CSHAKE_Get()

Description

Get data (squeeze).

Prototype

```
void CRYPTO_CSHAKE_Get(CRYPTO_CSHAKE_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to cSHAKE context.
pOutput	Pointer to object that receives the output.
OutputLen	Octet length of the output string.

11.3.1.10 CRYPTO_CSHAKE_Kill()

Description

Clear cSHAKE context.

Prototype

```
void CRYPTO_CSHAKE_Kill(CRYPTO_CSHAKE_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to cSHAKE context.

11.4 Keccak

Keccak is the building block used for the SHAKE and cSHAKE extendable output functions and the SHA-3 family of hash functions.

11.4.1 Type-safe API

Function	Description
Message functions	
CRYPTO_KECCAK_Init()	Initialize Keccak context.
CRYPTO_KECCAK_Add()	Add data to state (absorb).
CRYPTO_KECCAK_AddPadding()	Add final padding.
CRYPTO_KECCAK_Get()	Get output (squeeze).
CRYPTO_KECCAK_Kill()	Destroy a Keccak context.

11.4.1.1 CRYPTO_KECCAK_Init()

Description

Initialize Keccak context.

Prototype

```
void CRYPTO_KECCAK_Init(CRYPTO_KECCAK_CONTEXT * pSelf,  
                         unsigned Capacity);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
Capacity	Keccak capacity.

11.4.1.2 CRYPTO_KECCAK_Add()

Description

Add data to state (absorb).

Prototype

```
void CRYPTO_KECCAK_Add(          CRYPTO_KECCAK_CONTEXT * pSelf,
                           const U8           * pInput,
                           unsigned           InputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
pInput	Pointer to input data to absorb.
InputLen	Octet length of the input data.

11.4.1.3 CRYPTO_KECCAK_AddPadding()

Description

Add final padding.

Prototype

```
void CRYPTO_KECCAK_AddPadding(CRYPTO_KECCAK_CONTEXT * pSelf,  
                               U8 Padding);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
Padding	Padding to add.

11.4.1.4 CRYPTO_KECCAK_Get()

Description

Get output (squeeze).

Prototype

```
void CRYPTO_KECCAK_Get(CRYPTO_KECCAK_CONTEXT * pSelf,  
                        U8                      * pOutput,  
                        unsigned                 OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Keccak context.
pOutput	Pointer to object that receives the squeezed data.
OutputLen	Octet length of the receiving object.

11.4.1.5 CRYPTO_KECCAK_Kill()

Description

Destroy a Keccak context.

Prototype

```
void CRYPTO_KECCAK_Kill(CRYPTO_KECCAK_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to Keccak context.

Chapter 12

Asymmetric encryption (public key)

12.1 RSA

12.1.1 Introduction

A RSA key pair consists of a public key (which can be used for encryption) and a private key (which can be used for decryption).

Public RSA key

The public key has to be provided as object of type CRYPTO_RSA_PUBLIC_KEY to the API functions. It consists of two components:

- The modulus.
- The public exponent.

Both components are stored as multi precision integers in the CRYPTO_RSA_PUBLIC_KEY object. In most cases the public key is not available as object of this type in the application. Therefore the application has to load the public key into a CRYPTO_RSA_PUBLIC_KEY object before it can be used by cryptographic functions. This can be done using the multi precision integer function described in *Format conversion* on page 2608. Depending of the format the public key is available, an appropriate conversion function can be chosen.

Example

```
//  
// Public RSA key given as octet string in big endian byte order.  
//  
const U8 PublicExponent[] = { 0x01, 0x00, 0x01 };  
const U8 Modulus[] = { 0x8a, 0x8f, 0xa3, 0x9f, 0x9d, 0x71, ..., 0x29 };  
//  
// Public key object.  
//  
CRYPTO_RSA_PUBLIC_KEY PublicKey;  
//  
// Load public key.  
//  
CRYPTO_RSA_InitPublicKey(&PublicKey, &MemContext);  
if (CRYPTO_MPI_LoadBytes(&PublicKey.N, Modulus, sizeof(Modulus)) < 0 ||  
  
    CRYPTO_MPI_LoadBytes(&PublicKey.E, PublicExponent, sizeof(PublicExponent)) < 0) {  
    // error: Not enough memory  
}  
//  
// Public key can be used now.  
//  
r = CRYPTO_RSA_Encrypt(&PublicKey, pResult, ResultLen,  
                      pClearData, ClearDataLen, &MemContext);
```

For an explanation of the memory context MemContext refer to *Dynamic memory usage* on page 22.

Private RSA key

The private key has to be provided as object of type CRYPTO_RSA_PRIVATE_KEY to the API functions. It consists of the following components:

- Modulus.
- Private exponent (D).
- Prime factor P.
- Prime factor Q.
- First exponent for CRT, $dP := D \bmod (P-1)$
- Second exponent for CRT, $dQ := D \bmod (Q-1)$
- Coefficient for CRT, $U := Q^{-1} \bmod P$

Not all components are necessary for a private key operation. They are stored as multi precision integers in the CRYPTO_RSA_PRIVATE_KEY object. In most cases the private key is not available as object of this type in the application. Therefore the application has to load the private key into a CRYPTO_RSA_PRIVATE_KEY object before it can be used by cryptographic functions. This can be done using the multi precision integer function described in *Format conversion* on page 2608. Depending of the format the private key is available, an appropriate conversion function can be chosen.

Example

```
//  
// Private RSA key given as octet string in big endian byte order.  
//  
const U8 P[] = { 0xbb, 0x74, 0xf6, 0x08, 0x35, 0x5a, 0x87, ..., 0x77 };  
const U8 Q[] = { 0xbd, 0x39, 0xc0, 0x79, 0x9d, 0x9f, 0xa6, ..., 0x5F };  
const U8 dP[] = { 0x22, 0xf2, 0x89, 0x33, 0xba, 0x8e, 0xa8, ..., 0xdd };  
const U8 dQ[] = { 0x5f, 0x7d, 0xa1, 0x2d, 0x61, 0x93, 0xa9, ..., 0x18 };  
const U8 U[] = { 0x2c, 0x13, 0x24, 0x9a, 0xef, 0x34, 0xfd, ..., 0x1f };  
//  
// Private key object.  
//  
CRYPTO_RSA_PRIVATE_KEY PrivateKey;  
//  
// Load private key.  
//  
CRYPTO_RSA_InitPrivateKey(&PrivateKey, &MemContext);  
if (CRYPTO_MPI_LoadBytes(&PrivateKey.P, P, sizeof(P)) < 0 ||  
    CRYPTO_MPI_LoadBytes(&PrivateKey.Q, Q, sizeof(Q)) < 0 ||  
    CRYPTO_MPI_LoadBytes(&PrivateKey.DP, dP, sizeof(dP)) < 0 ||  
    CRYPTO_MPI_LoadBytes(&PrivateKey.DQ, dQ, sizeof(dQ)) < 0 ||  
    CRYPTO_MPI_LoadBytes(&PrivateKey.QInv, U, sizeof(U)) < 0) {  
    // error: Not enough memory  
}  
//  
// Private key can be used now.  
//  
r = CRYPTO_RSA_Decrypt(&PrivateKey, pResult, ResultLen,  
                      pCipherData, CipherDataLen, &MemContext);
```

For an explanation of the memory context MemContext refer to *Dynamic memory usage* on page 22.

12.1.2 Data types

Type	Description
CRYPTO_RSA_PRIVATE_KEY	RSA private key data.
CRYPTO_RSA_PUBLIC_KEY	RSA public key data.

12.1.2.1 CRYPTO_RSA_PRIVATE_KEY

Description

RSA private key data.

Type definition

```
typedef struct {
    CRYPTO_MPI D;
    CRYPTO_MPI P;
    CRYPTO_MPI Q;
    CRYPTO_MPI DP;
    CRYPTO_MPI DQ;
    CRYPTO_MPI QInv;
    CRYPTO_MPI N;
    CRYPTO_MPI E;
} CRYPTO_RSA_PRIVATE_KEY;
```

Structure members

Member	Description
D	Decryption exponent (non-CRT form).
P	Factor p of the public modulus.
Q	Factor q of the public modulus.
DP	$d \bmod (p-1)$
DQ	$d \bmod (q-1)$
QInv	$q^{-1} \bmod p$, i.e. ModInv(q, p)
N	Public modulus (non-CRT form).
E	Encryption exponent.

12.1.2.2 CRYPTO_RSA_PUBLIC_KEY

Description

RSA public key data.

Type definition

```
typedef struct {
    CRYPTO_MPI   N;
    CRYPTO_MPI   E;
} CRYPTO_RSA_PUBLIC_KEY;
```

Structure members

Member	Description
N	Public modulus, pq
E	Public encryption exponent

12.1.3 Management functions

Function	Description
<code>CRYPTO_RSA_InitPublicKey()</code>	Initialize RSA public key object before use.
<code>CRYPTO_RSA_InitPrivateKey()</code>	Initialize RSA private key object before use.
<code>CRYPTO_RSA_KillPublicKey()</code>	Zero all data relating to the public key and reclaim storage.
<code>CRYPTO_RSA_KillPrivateKey()</code>	Zero all data relating to the private key and reclaim storage.

12.1.3.1 CRYPTO_RSA_InitPrivateKey()

Description

Initialize RSA private key object before use. The function creates an empty private key object.

Prototype

```
void CRYPTO_RSA_InitPrivateKey(CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to initialize.
pMem	Allocator to use for expanding components of a private key.

12.1.3.2 CRYPTO_RSA_InitPublicKey()

Description

Initialize RSA public key object before use. The function creates an empty private key object.

Prototype

```
void CRYPTO_RSA_InitPublicKey(CRYPTO_RSA_PUBLIC_KEY * pSelf,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to initialize.
pMem	Allocator to use for expanding components of a public key.

12.1.3.3 CRYPTO_RSA_KillPrivateKey()

Description

Zero all data relating to the private key and reclaim storage.

Prototype

```
void CRYPTO_RSA_KillPrivateKey(CRYPTO_RSA_PRIVATE_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Private key to burn.

12.1.3.4 CRYPTO_RSA_KillPublicKey()

Description

Zero all data relating to the public key and reclaim storage.

Prototype

```
void CRYPTO_RSA_KillPublicKey(CRYPTO_RSA_PUBLIC_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Public key to burn.

12.1.4 Encryption functions

Function	Description
<code>CRYPTO_RSA_EncryptMPIToMPI()</code>	Encrypts a plaintext MPI to a ciphertext MPI using a public key.
<code>CRYPTO_RSA_Encrypt()</code>	Encrypts the plaintext to the ciphertext using a public key.
<code>CRYPTO_RSA_EncryptMPI()</code>	Encrypts the text using a public key.

12.1.4.1 CRYPTO_RSA_Encrypt()

Description

Encrypts the plaintext to the ciphertext using a public key.

Prototype

```
int CRYPTO_RSA_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,  
                      U8 * pOutput,  
                      unsigned OutputLen,  
                      const U8 * pInput,  
                      unsigned InputLen,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to public key for encryption.
pOutput	Pointer to object that receives the ciphered message.
OutputLen	Octet length of the receiving object.
pInput	Pointer to octet string containing the plaintext message.
InputLen	Octet length of the plaintext message.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.4.2 CRYPTO_RSA_EncryptMPI()

Description

Encrypts the text using a public key. CRYPTO_RSA_Encrypt() assumes that original plaintext is less than the modulus.

Prototype

```
int CRYPTO_RSA_EncryptMPI(const CRYPTO_RSA_PUBLIC_KEY * pSelf,  
                           CRYPTO_MPI           * pText,  
                           CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pText	Plaintext MPI on entry, ciphered MPI on return.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.4.3 CRYPTO_RSA_EncryptMPIToMPI()

Description

Encrypts a plaintext MPI to a ciphertext MPI using a public key. `CRYPTO_RSA_EncryptMPI()` assumes that plaintext is less than the modulus.

Prototype

```
int CRYPTO_RSA_EncryptMPIToMPI(const CRYPTO_RSA_PUBLIC_KEY * pSelf,  
                                CRYPTO_MPI           * pOutput,  
                                const CRYPTO_MPI       * pInput,  
                                CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
<code>pSelf</code>	Public key to encrypt with.
<code>pOutput</code>	Ciphered MPI.
<code>pInput</code>	Plaintext MPI.
<code>pMem</code>	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.5 Decryption functions

Function	Description
<code>CRYPTO_RSA_DecryptMPIToMPI()</code>	Decrypts a ciphertext MPI to a plaintext MPI using a private key.
<code>CRYPTO_RSA_DecryptMPI()</code>	Decrypts the ciphertext to the plaintext using a private key.
<code>CRYPTO_RSA_Decrypt()</code>	Decrypts a ciphertext message to plaintext message using a private key.
<code>CRYPTO_RSA_DecryptMPINonCRT()</code>	Decrypts the ciphertext to the plaintext using a private key and the standard decryption exponent (rather than CRT form).

12.1.5.1 CRYPTO_RSA_Decrypt()

Description

Decrypts a ciphertext message to plaintext message using a private key.

Prototype

```
int CRYPTO_RSA_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                      U8 * pOutput,  
                      unsigned OutputLen,  
                      const U8 * pInput,  
                      unsigned InputLen,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the object that receives the decrypted message.
pInput	Pointer to octet string that contains the ciphered message.
InputLen	Octet length of the ciphered message.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.5.2 CRYPTO_RSA_DecryptMPI()

Description

Decrypts the ciphertext to the plaintext using a private key.

Prototype

```
int CRYPTO_RSA_DecryptMPI(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                           CRYPTO_MPI           * pText,  
                           CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to decrypt with.
pText	Ciphered MPI on entry, plaintext MPI on return.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.5.3 CRYPTO_RSA_DecryptMPINonCRT()

Description

Decrypts the ciphertext to the plaintext using a private key and the standard decryption exponent (rather than CRT form).

Prototype

```
int CRYPTO_RSA_DecryptMPINonCRT(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                                 CRYPTO_MPI          * pText,  
                                 CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
pSelf	Private key.
pText	Data to be decrypted (to itself).
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.5.4 CRYPTO_RSA_DecryptMPIToMPI()

Description

Decrypts a ciphertext MPI to a plaintext MPI using a private key.

Prototype

```
int CRYPTO_RSA_DecryptMPIToMPI(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                                CRYPTO_MPI           * pOutput,  
                                const CRYPTO_MPI      * pInput,  
                                CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Decrypted MPI (aka plaintext).
pInput	Ciphered MPI (aka ciphertext).
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.6 Utility functions

Function	Description
<code>CRYPTO_RSA_CalcDecryptExponent()</code>	Given two primes in the private key and an exponent in the public key, compute the decryption exponent and the CRT form of the private key.
<code>CRYPTO_RSA_ConstructKeys()</code>	initialize the private and public key structures given two primes and a public exponent.
<code>CRYPTO_RSA_RecoverModulus()</code>	Computes the modulus pq into pModulus from the modulus factors P and Q held in the private key.
<code>CRYPTO_RSA_ModulusBits()</code>	Computes the number of modulus bits from the modulus factors P and Q held in the private key.
<code>CRYPTO_RSA_ModulusBytes()</code>	Computes the number of modulus bytes from the modulus factors P and Q held in the private key.
<code>CRYPTO_RSA_ModulusBytes_ASN1()</code>	Inquire number of bytes to encode the modulus as an ASN.1 integer.
<code>CRYPTO_RSA_IsConsistentPair()</code>	Predicate which determines whether the parameters held in the private and public keys of an RSA key pair are consistent.

12.1.6.1 CRYPTO_RSA_CalcDecryptExponent()

Description

Given two primes in the private key and an exponent in the public key, compute the decryption exponent and the CRT form of the private key.

Prototype

```
int CRYPTO_RSA_CalcDecryptExponent(          CRYPTO_RSA_PRIVATE_KEY * pPrivateKey,
                                         const CRYPTO_RSA_PUBLIC_KEY * pPublicKey,
                                         CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to object that receives the private key.
pPublicKey	Pointer to RSA public key.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

12.1.6.2 CRYPTO_RSA_ConstructKeys()

Description

initialize the private and public key structures given two primes and a public exponent. This function does not check that the parameters make a valid key pair, you can use CRYPTO_RSA_IsConsistentPair() for that.

Prototype

```
int CRYPTO_RSA_ConstructKeys( CRYPTO_RSA_PRIVATE_KEY * pPrivateKey,  
                             CRYPTO_RSA_PUBLIC_KEY * pPublicKey,  
                             CRYPTO_MPI * pP,  
                             CRYPTO_MPI * pQ,  
                             const CRYPTO_MPI * pExponent,  
                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Private key to construct.
pPublicKey	Public key to construct.
pP	First prime factor of the modulus.
pQ	Second prime factor of the modulus.
pExponent	Public encryption exponent.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Success.

12.1.6.3 CRYPTO_RSA_IsConsistentPair()

Description

Predicate which determines whether the parameters held in the private and public keys of an RSA key pair are consistent.

Prototype

```
int CRYPTO_RSA_IsConsistentPair(const CRYPTO_RSA_PUBLIC_KEY * pPublicKey,  
                                const CRYPTO_RSA_PRIVATE_KEY * pPrivateKey,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPublicKey	Public key to validate.
pPrivateKey	Private key to validate.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing success, but keys are not consistent.
- > 0 Processing success, keys are consistent.

12.1.6.4 CRYPTO_RSA_ModulusBits()

Description

Computes the number of modulus bits from the modulus factors P and Q held in the private key.

Prototype

```
int CRYPTO_RSA_ModulusBits(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                           CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Private key.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 The number of bits in the modulus.

12.1.6.5 CRYPTO_RSA_ModulusBytes()

Description

Computes the number of modulus bytes from the modulus factors P and Q held in the private key.

Prototype

```
int CRYPTO_RSA_ModulusBytes(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                           CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Private key.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 The number of bytes in the modulus.

Additional information

This function returns the minimum number of bytes required to encode the modulus and considers the modulus unsigned. Therefore, the most significant byte of the encoded form is allowed to have its most significant bit set. Should you need to compute the number of bytes to encode a non-negative ASN.1 modulus, use `CRYPTO_RSA_ModulusBytesN()`.

12.1.6.6 CRYPTO_RSA_ModulusBytes_ASN1()

Description

Inquire number of bytes to encode the modulus as an ASN.1 integer.

Prototype

```
int CRYPTO_RSA_ModulusBytes_ASN1(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                                 CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pSelf	Private key.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 The number of bytes in the modulus.

Additional information

This function returns the minimum number of bytes required to encode the modulus in ASN.1 form where the most significant byte of the encoded form has its most significant bit set to zero.

The number of modulus bits is computed from the modulus factors P and Q held in the private key.

12.1.6.7 CRYPTO_RSA_RecoverModulus()

Description

Computes the modulus pq into `pModulus` from the modulus factors P and Q held in the private key.

Prototype

```
int CRYPTO_RSA_RecoverModulus(const CRYPTO_RSA_PRIVATE_KEY * pSelf,  
                               CRYPTO_MPI           * pModulus,  
                               CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
<code>pSelf</code>	Private key.
<code>pModulus</code>	Modulus calculated from private key.
<code>pMem</code>	Allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Chapter 13

Digital signatures

13.1 RSA

13.1.1 Introduction

A RSA key pair consists of a private key (which can be used to create signatures) and a public key (which can be used to verify signatures).

Public RSA key

The public key has to be provided as object of type CRYPTO_RSA_PUBLIC_KEY to the API functions. It consists of two components:

- The modulus.
- The public exponent.

Both components are stored as multi precision integers in the CRYPTO_RSA_PUBLIC_KEY object. In most cases the public key is not available as object of this type in the application. Therefore the application has to load the public key into a CRYPTO_RSA_PUBLIC_KEY object before it can be used by cryptographic functions. This can be done using the multi precision integer function described in *Format conversion* on page 2608. Depending of the format the public key is available, an appropriate conversion function can be chosen.

Example

```
//  
// Public RSA key given as octet string in big endian byte order.  
//  
const U8 PublicExponent[] = { 0x01, 0x00, 0x01 };  
const U8 Modulus[] = { 0x8a, 0x8f, 0xa3, 0x9f, 0x9d, 0x71, ..., 0x29 };  
//  
// Public key object.  
//  
CRYPTO_RSA_PUBLIC_KEY PublicKey;  
//  
// Load public key.  
//  
CRYPTO_RSA_InitPublicKey(&PublicKey, &MemContext);  
if (CRYPTO_MPI_LoadBytes(&PublicKey.N, Modulus, sizeof(Modulus)) < 0 ||  
    CRYPTO_MPI_LoadBytes(&PublicKey.E, PublicExponent, sizeof(PublicExponent)) < 0) {  
    // error: Not enough memory  
}  
//  
// Public key can be used now.  
//  
r = CRYPTO_RSASSA_PKCS1_SHA1_Verify(&PublicKey, pMessage, MessageLen, NULL, 0,  
                                    pSignature, SignatureLen, &MemContext);
```

For an explanation of the memory context MemContext refer to *Dynamic memory usage* on page 22.

Private RSA key

The private key has to be provided as object of type CRYPTO_RSA_PRIVATE_KEY to the API functions. It consists of the following components:

- Modulus.
- Private exponent (D).
- Prime factor P.
- Prime factor Q.
- First exponent for CRT, $dP := D \bmod (P-1)$
- Second exponent for CRT, $dQ := D \bmod (Q-1)$
- Coefficient for CRT, $U := Q^{-1} \bmod P$

Not all components are necessary for a private key operation. They are stored as multi precision integers in the CRYPTO_RSA_PRIVATE_KEY object. In most cases the private key is not available as object of this type in the application. Therefore the application has to load the private key into a CRYPTO_RSA_PRIVATE_KEY object before it can be used by cryptographic functions. This can be done using the multi precision integer function described in *Format conversion* on page 2608. Depending of the format the private key is available, an appropriate conversion function can be chosen.

Example

```

//  

// Private RSA key given as octet string in big endian byte order.  

//  

const U8 P[] = { 0xbb, 0x74, 0xf6, 0x08, 0x35, 0x5a, 0x87, ..., 0x77 };  

const U8 Q[] = { 0xbd, 0x39, 0xc0, 0x79, 0x9d, 0x9f, 0xa6, ..., 0x5F };  

const U8 dP[] = { 0x22, 0xf2, 0x89, 0x33, 0xba, 0x8e, 0xa8, ..., 0xdd };  

const U8 dQ[] = { 0x5f, 0x7d, 0xa1, 0x2d, 0x61, 0x93, 0xa9, ..., 0x18 };  

const U8 U[] = { 0x2c, 0x13, 0x24, 0x9a, 0xef, 0x34, 0xfd, ..., 0x1f };  

//  

// Private key object.  

//  

CRYPTO_RSA_PRIVATE_KEY PrivateKey;  

//  

// Load private key.  

//  

CRYPTO_RSA_InitPrivateKey(&PrivateKey, &MemContext);  

if (CRYPTO_MPI_LoadBytes(&PrivateKey.P, P, sizeof(P)) < 0 ||  

    CRYPTO_MPI_LoadBytes(&PrivateKey.Q, Q, sizeof(Q)) < 0 ||  

    CRYPTO_MPI_LoadBytes(&PrivateKey.DP, dP, sizeof(dP)) < 0 ||  

    CRYPTO_MPI_LoadBytes(&PrivateKey.DQ, dQ, sizeof(dQ)) < 0 ||  

    CRYPTO_MPI_LoadBytes(&PrivateKey.QInv, U, sizeof(U)) < 0) {  

    // error: Not enough memory  

}  

//  

// Private key can be used now.  

//  

r = CRYPTO_RSASSA_PKCS1_SHA1_Sign(&PrivateKey, pMessage, MessageLen, 0,  

                                    pSignature, SignatureLen, &MemContext);
```

For an explanation of the memory context MemContext refer to *Dynamic memory usage* on page 22.

13.1.2 Key generation

The following table lists the RSA PKCS#1 type-safe key generation functions.

Function	Description
<code>CRYPTO_RSA_P1363_GenKeys()</code>	Generate an RSA key pair into the private and public key structures.
<code>CRYPTO_RSA_FIPS186_GenKeys()</code>	Generate a public and private key pair.
<code>CRYPTO_RSA_FIPS186_GenPrime()</code>	Generate a Shawe-Taylor provable prime of arbitrary size as per FIPS 186-4 section C.10 with N1 = 1 and N2 = 2.
<code>CRYPTO_RSA_FIPS186_GenPrimePair()</code>	Generate a pair of provable prime of arbitrary size as per FIPS 186-4 section B.3.2, "Generation of Random Primes that are Provably Prime".
<code>CRYPTO_RSA_FIPS186_ValidateParaSize()</code>	Validate that the modulus size L is acceptable by the FIPS 186-4 standard.

13.1.2.1 CRYPTO_RSA_P1363_GenKeys()

Description

Generate an RSA key pair into the private and public key structures. The generated modulus is `ModulusBits` in size. If you call `Crypto_RSA_GenerateKeys()` with an exponent that is null or zero, `Crypto_RSA_GenerateKeys()` will choose an appropriate, small public exponent for you. If you call `Crypto_RSA_GenerateKeys()` with a chosen (fixed) public exponent, that exponent is assigned to the public key pair.

Prototype

```
int CRYPTO_RSA_P1363_GenKeys(          CRYPTO_RSA_PRIVATE_KEY * pPrivateKey,
                                         CRYPTO_RSA_PUBLIC_KEY * pPublicKey,
                                         unsigned                ModulusBits,
                                         const CRYPTO_MPI        * pExponent,
                                         CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
<code>pPrivateKey</code>	Generated private key.
<code>pPublicKey</code>	Generated public key.
<code>ModulusBits</code>	Size of the public modulus, in bits.
<code>pExponent</code>	Public encryption exponent.
<code>pMem</code>	Allocator to use for temporary storage.

Return value

- < 0 Error generating keys.
- ≥ 0 Key generation successful.

13.1.2.2 CRYPTO_RSA_FIPS186_GenKeys()

Description

Generate a public and private key pair.

Prototype

```
int CRYPTO_RSA_FIPS186_GenKeys( CRYPTO_RSA_PRIVATE_KEY * pPrivateKey,
                                CRYPTO_RSA_PUBLIC_KEY * pPublicKey,
                                U8                         * pSeed,
                                unsigned                     SeedLen,
                                unsigned                     ModulusBits,
                                const CRYPTO_MPI           * pExponent,
                                const CRYPTO_FIPS186_PRIMEGEN_API * pPrimeAPI,
                                CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Generated private key.
pPublicKey	Generated public key.
pSeed	Initial seed, zeroed upon return.
SeedLen	Number of bytes in the seed array.
ModulusBits	Size of the public modulus, in bits.
pExponent	Public encryption exponent.
pPrimeAPI	Pointer to prime generation API.
pMem	Allocator to use for temporary storage.

Return value

- ≤ 0 Failed to generate a key pair.
- > 0 Successful generation of a proven prime key pair.

13.1.2.3 CRYPTO_RSA_FIPS186_GenPrime()

Description

Generate a Shawe-Taylor provable prime of arbitrary size as per FIPS 186-4 section C.10 with N1 = 1 and N2 = 2.

Prototype

```
int CRYPTO_RSA_FIPS186_GenPrime( CRYPTO_MPI * pPrime,
                                  unsigned PrimeLen,
                                  U8          * pSeed,
                                  unsigned SeedLen,
                                  const CRYPTO_MPI * pE,
                                  const CRYPTO_FIPS186_PRIMEGEN_API * pPrimeAPI,
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrime	Pointer to MPI that receives the generated prime.
PrimeLen	Number of bits in the generated prime.
pSeed	Initial seed, updated upon return for subsequent calls to generate additional random numbers with updated seed.
SeedLen	Octet length of the seed.
pE	Public exponent that must be coprime to the generated prime, minus one.
pPrimeAPI	Pointer to prime generation API.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing successful but no prime generated.
- > 0 Processing successful with prime generated.

13.1.2.4 CRYPTO_RSA_FIPS186_GenPrimePair()

Description

Generate a pair of provable prime of arbitrary size as per FIPS 186-4 section B.3.2, "Generation of Random Primes that are Provably Prime".

Prototype

```
int CRYPTO_RSA_FIPS186_GenPrimePair( CRYPTO_MPI * pP,
                                      CRYPTO_MPI * pQ,
                                      U8          * pSeed,
                                      unsigned    SeedLen,
                                      unsigned    ModulusLen,
                                      const CRYPTO_MPI * pE,
                                      const CRYPTO_FIPS186_PRIMEGEN_API * pPrimeAPI,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pP	Generated prime #1, P.
pQ	Generated prime #2, Q.
pSeed	Initial seed, zeroed upon return.
SeedLen	Octet length of the seed.
ModulusLen	Number of bits in product of the primes P and Q, i.e. the size of a public modulus in bits.
pE	Public exponent that must be coprime to P-1 and Q-1.
pPrimeAPI	Pointer to prime generation API.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Failure to generate a prime pair
- ≥ 0 Successful generation of a proven prime pair.

13.1.2.5 CRYPTO_RSA_FIPS186_ValidateParaSize()

Description

Validate that the modulus size [L](#) is acceptable by the FIPS 186-4 standard.

Prototype

```
int CRYPTO_RSA_FIPS186_ValidateParaSize(unsigned L);
```

Parameters

Parameter	Description
L	Length of modulus to validate, in bits.

Return value

- = 0 Parameters are not acceptable.
- ≠ 0 Parameters are valid.

13.1.3 RSASSA-PKCS1 message sign and verify

The following table lists the RSASSA-PKCS#1 type-safe message sign and verify API functions.

Function	Description
Sign message	
<code>CRYPTO_RSASSA_PKCS1_SHA1_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA224_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA256_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA384_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_224_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_256_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_224_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_256_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_384_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_512_Sign()</code>	Sign hashed message according to PKCS#1 version 1.5.
Verify message	
<code>CRYPTO_RSASSA_PKCS1_SHA1_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA224_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA256_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA384_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_224_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_256_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_224_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_256_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_384_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_512_Verify()</code>	Sign hashed message according to PKCS#1 version 1.5.

13.1.3.1 CRYPTO_RSASSA_PKCS1_SHA1_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA1_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8                           * pMessage,
                                    unsigned                         MessageLen,
                                    const U8                           * pSalt,
                                    unsigned                         SaltLen,
                                    U8                               * pSignature,
                                    unsigned                         SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA1 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.2 CRYPTO_RSASSA_PKCS1_SHA1_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA1_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                     const U8 * pMessage,
                                     unsigned U8 MessageLen,
                                     unsigned U8 * pSalt,
                                     const U8 SaltLen,
                                     unsigned * pSignature,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA1 as the hash function.

13.1.3.3 CRYPTO_RSASSA_PKCS1_SHA224_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA224_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.4 CRYPTO_RSASSA_PKCS1_SHA224_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA224_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8               * pMessage,
                                       unsigned             MessageLen,
                                       U8                  * pSalt,
                                       unsigned             SaltLen,
                                       const U8              * pSignature,
                                       unsigned            SignatureLen,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA224 as the hash function.

13.1.3.5 CRYPTO_RSASSA_PKCS1_SHA256_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA256_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.6 CRYPTO_RSASSA_PKCS1_SHA256_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA256_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8               * pMessage,
                                       unsigned             MessageLen,
                                       U8                  * pSalt,
                                       unsigned             SaltLen,
                                       const U8              * pSignature,
                                       unsigned            SignatureLen,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA256 as the hash function.

13.1.3.7 CRYPTO_RSASSA_PKCS1_SHA384_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA384_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA384 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.8 CRYPTO_RSASSA_PKCS1_SHA384_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA384_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8               * pMessage,
                                       unsigned             MessageLen,
                                       U8                  * pSalt,
                                       unsigned             SaltLen,
                                       const U8              * pSignature,
                                       unsigned            SignatureLen,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA384 as the hash function.

13.1.3.9 CRYPTO_RSASSA_PKCS1_SHA512_224_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_224_Sign
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessage,
     unsigned                     MessageLen,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512_224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.10 CRYPTO_RSASSA_PKCS1_SHA512_224_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_224_Verify
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessage,
     unsigned                 MessageLen,
     U8                      * pSalt,
     unsigned                 SaltLen,
     const U8                  * pSignature,
     unsigned                 SignatureLen,
     CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512_224 as the hash function.

13.1.3.11 CRYPTO_RSASSA_PKCS1_SHA512_256_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_256_Sign
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessage,
     unsigned                     MessageLen,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512_256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.12 CRYPTO_RSASSA_PKCS1_SHA512_256_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_256_Verify
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessage,
     unsigned                 MessageLen,
     U8                      * pSalt,
     unsigned                 SaltLen,
     const U8                  * pSignature,
     unsigned                 SignatureLen,
     CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512_256 as the hash function.

13.1.3.13 CRYPTO_RSASSA_PKCS1_SHA512_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.14 CRYPTO_RSASSA_PKCS1_SHA512_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8 * pMessage,
                                       unsigned U8 MessageLen,
                                       unsigned U8 * pSalt,
                                       const U8 SaltLen,
                                       unsigned * pSignature,
                                       CRYPTO_MEM_CONTEXT * pSignatureLen,
                                       * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512 as the hash function.

13.1.3.15 CRYPTO_RSASSA_PKCS1_SHA3_224_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_224_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.16 CRYPTO_RSASSA_PKCS1_SHA3_224_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_224_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned U8 MessageLen,
                                         unsigned U8 * pSalt,
                                         const U8 SaltLen,
                                         unsigned * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_224 as the hash function.

13.1.3.17 CRYPTO_RSASSA_PKCS1_SHA3_256_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_256_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.18 CRYPTO_RSASSA_PKCS1_SHA3_256_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_256_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         U8 * pSalt,
                                         unsigned SaltLen,
                                         const U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_256 as the hash function.

13.1.3.19 CRYPTO_RSASSA_PKCS1_SHA3_384_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_384_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_384 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.20 CRYPTO_RSASSA_PKCS1_SHA3_384_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_384_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned U8 MessageLen,
                                         unsigned U8 * pSalt,
                                         const U8 SaltLen,
                                         unsigned * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_384 as the hash function.

13.1.3.21 CRYPTO_RSASSA_PKCS1_SHA3_512_Sign()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_512_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessage	Message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_512 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.3.22 CRYPTO_RSASSA_PKCS1_SHA3_512_Verify()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_512_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned U8 MessageLen,
                                         unsigned U8 * pSalt,
                                         const U8 SaltLen,
                                         unsigned * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_512 as the hash function.

13.1.4 RSASSA-PKCS1 digest sign and verify

The following table lists the RSASSA-PKCS#1 type-safe digest sign and verify API functions.

Function	Description
Sign digest	
<code>CRYPTO_RSASSA_PKCS1_SHA1_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA224_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA256_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA384_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_224_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_256_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_224_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_256_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_384_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_512_SignDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
Verify digest	
<code>CRYPTO_RSASSA_PKCS1_SHA1_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA224_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA256_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA384_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_224_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA512_256_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_224_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_256_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_384_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.
<code>CRYPTO_RSASSA_PKCS1_SHA3_512_VerifyDigest()</code>	Sign hashed message according to PKCS#1 version 1.5.

Function	Description
Low-level functions	
<code>CRYPTO_RSASSA_PKCS1_SignDigest()</code>	Sign data using RSASSA-PKCS1-v1_5.
<code>CRYPTO_RSA_PKCS1_Unwrap()</code>	Decrypt a signature according to PKCS#1 version 1.5.

13.1.4.1 CRYPTO_RSASSA_PKCS1_SHA1_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA1_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA1 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.2 CRYPTO_RSASSA_PKCS1_SHA1_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA1_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA1 as the hash function.

13.1.4.3 CRYPTO_RSASSA_PKCS1_SHA224_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.4 CRYPTO_RSASSA_PKCS1_SHA224_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA224 as the hash function.

13.1.4.5 CRYPTO_RSASSA_PKCS1_SHA256_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.6 CRYPTO_RSASSA_PKCS1_SHA256_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA256 as the hash function.

13.1.4.7 CRYPTO_RSASSA_PKCS1_SHA384_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA384_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA384 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.8 CRYPTO_RSASSA_PKCS1_SHA384_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA384_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA384 as the hash function.

13.1.4.9 CRYPTO_RSASSA_PKCS1_SHA512_224_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512_224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.10 CRYPTO_RSASSA_PKCS1_SHA512_224_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512_224 as the hash function.

13.1.4.11 CRYPTO_RSASSA_PKCS1_SHA512_256_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512_256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.12 CRYPTO_RSASSA_PKCS1_SHA512_256_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512_256 as the hash function.

13.1.4.13 CRYPTO_RSASSA_PKCS1_SHA512_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA512 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.14 CRYPTO_RSASSA_PKCS1_SHA512_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA512_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA512 as the hash function.

13.1.4.15 CRYPTO_RSASSA_PKCS1_SHA3_224_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_224 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.16 CRYPTO_RSASSA_PKCS1_SHA3_224_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_224 as the hash function.

13.1.4.17 CRYPTO_RSASSA_PKCS1_SHA3_256_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_256 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.18 CRYPTO_RSASSA_PKCS1_SHA3_256_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_256 as the hash function.

13.1.4.19 CRYPTO_RSASSA_PKCS1_SHA3_384_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_384_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_384 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.20 CRYPTO_RSASSA_PKCS1_SHA3_384_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_384_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_384 as the hash function.

13.1.4.21 CRYPTO_RSASSA_PKCS1_SHA3_512_SignDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_512_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pMessageHash,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Private key for encryption.
pMessageHash	Digest to sign.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but signature failure (signature buffer too small, salt given).
- > 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-SIGN using EMSA-PKCS1-v1_5-ENCODE according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to *pInput and the ciphertext C is equivalent to *pOutput.

This implementation uses SHA3_512 as the hash function.

For reference, see PKCS #1 v2.2 section 9.2, EMCSA_PKCS1-v1_5.

13.1.4.22 CRYPTO_RSASSA_PKCS1_SHA3_512_VerifyDigest()

Description

Sign hashed message according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SHA3_512_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pMessageHash,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for verification.
pMessageHash	Digest to verify.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Size of salt octet string in bytes.
pSignature	Signature of message.
SignatureLen	Size of signature buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

Additional information

The RSASSA-PKCS1-v1_5 signature scheme does not provide the capability to add and recover a salt from the signature. Therefore, this function zeros the salt octet string. This decision is taken such that this function prototype exactly matches the corresponding prototype for the RSASSA-PSS signature scheme and they can, therefore, be used somewhat interchangeably in source code.

This implementation uses SHA3_512 as the hash function.

13.1.4.23 CRYPTO_RSASSA_PKCS1_SignDigest()

Description

Sign data using RSASSA-PKCS1-v1_5.

Prototype

```
int CRYPTO_RSASSA_PKCS1_SignDigest(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8 * pDigest,
                                    unsigned DigestLen,
                                    U8 * pSignature,
                                    unsigned SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for signing.
pDigest	Pointer to digest to sign, typically a DigestInfo octet string.
DigestLen	Octet length of digest octet string.
pSignature	Pointer to object that receives the signed digest.
SignatureLen	Octet length of the signed digest object.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Encryption successful, number of bytes in encrypted message.

Additional information

This is an implementation of RSASSA-PKCS1-v1_5-Sign using EMSA-PKCS1-v1_5-Encode according to PKCS #1 and RFC 2437. In this instance, M of RFC 2437 is equivalent to pInput[] and the ciphertext C is equivalent to pOutput[].

13.1.4.24 CRYPTO_RSA_PKCS1_Unwrap()

Description

Decrypt a signature according to PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSA_PKCS1_Unwrap(const CRYPTO_RSA_PUBLIC_KEY * pSelf,  
                           const U8 * pInput,  
                           unsigned InputLen,  
                           U8 * pOutput,  
                           unsigned OutputLen,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key for decryption.
pInput	Message to decrypt.
InputLen	Octet length of message to decrypt.
pOutput	Decrypted message buffer.
OutputLen	Octet length of decrypted message buffer.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Decryption successful, number of bytes in decrypted message.

13.1.5 RSASSA-PSS message sign and verify

The following table lists the RSASSA-PSS type-safe message sign and verify API functions.

Function	Description
Sign message	
<code>CRYPTO_RSASSA_PSS_SHA1_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA224_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA256_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA384_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_224_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_256_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_224_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_256_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_384_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_512_Sign()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
Verify message	
<code>CRYPTO_RSASSA_PSS_SHA1_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA224_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA256_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA384_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_224_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_256_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_224_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_256_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_384_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_512_Verify()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

13.1.5.1 CRYPTO_RSASSA_PSS_SHA1_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA1_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                  const U8 * pMessage,
                                  unsigned MessageLen,
                                  const U8 * pSalt,
                                  unsigned SaltLen,
                                  U8 * pSignature,
                                  unsigned SignatureLen,
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.2 CRYPTO_RSASSA_PSS_SHA1_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA1_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                    const U8                         * pMessage,
                                    unsigned                           MessageLen,
                                    U8                               * pSalt,
                                    unsigned                           SaltLen,
                                    const U8                         * pSignature,
                                    unsigned                          SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.3 CRYPTO_RSASSA_PSS_SHA224_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA224_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8                           * pMessage,
                                    unsigned                         MessageLen,
                                    const U8                           * pSalt,
                                    unsigned                         SaltLen,
                                    U8                               * pSignature,
                                    unsigned                         SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.4 CRYPTO_RSASSA_PSS_SHA224_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA224_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      U8                               * pSalt,
                                      unsigned                         SaltLen,
                                      const U8                          * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.5 CRYPTO_RSASSA_PSS_SHA256_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA256_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8                           * pMessage,
                                    unsigned                         MessageLen,
                                    const U8                           * pSalt,
                                    unsigned                         SaltLen,
                                    U8                               * pSignature,
                                    unsigned                         SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.6 CRYPTO_RSASSA_PSS_SHA256_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA256_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      U8                               * pSalt,
                                      unsigned                         SaltLen,
                                      const U8                          * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.7 CRYPTO_RSASSA_PSS_SHA384_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA384_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8                           * pMessage,
                                    unsigned                         MessageLen,
                                    const U8                           * pSalt,
                                    unsigned                         SaltLen,
                                    U8                               * pSignature,
                                    unsigned                         SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.8 CRYPTO_RSASSA_PSS_SHA384_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA384_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      U8                               * pSalt,
                                      unsigned                         SaltLen,
                                      const U8                          * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.9 CRYPTO_RSASSA_PSS_SHA512_224_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_224_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.10 CRYPTO_RSASSA_PSS_SHA512_224_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_224_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned U8 MessageLen,
                                         unsigned U8 * pSalt,
                                         const U8 SaltLen,
                                         unsigned * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.11 CRYPTO_RSASSA_PSS_SHA512_256_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_256_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         const U8 * pSalt,
                                         unsigned SaltLen,
                                         U8 * pSignature,
                                         unsigned SignatureLen,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.12 CRYPTO_RSASSA_PSS_SHA512_256_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_256_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8 * pMessage,
                                         unsigned U8 MessageLen,
                                         unsigned U8 * pSalt,
                                         const U8 SaltLen,
                                         unsigned * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.13 CRYPTO_RSASSA_PSS_SHA512_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                    const U8                           * pMessage,
                                    unsigned                         MessageLen,
                                    const U8                           * pSalt,
                                    unsigned                         SaltLen,
                                    U8                               * pSignature,
                                    unsigned                         SignatureLen,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.14 CRYPTO_RSASSA_PSS_SHA512_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      U8                               * pSalt,
                                      unsigned                         SaltLen,
                                      const U8                          * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.15 CRYPTO_RSASSA_PSS_SHA3_224_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_224_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.16 CRYPTO_RSASSA_PSS_SHA3_224_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_224_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8 * pMessage,
                                       unsigned U8 MessageLen,
                                       unsigned U8 * pSalt,
                                       const U8 SaltLen,
                                       unsigned * pSignature,
                                       CRYPTO_MEM_CONTEXT * pSignatureLen,
                                       * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.17 CRYPTO_RSASSA_PSS_SHA3_256_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_256_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.18 CRYPTO_RSASSA_PSS_SHA3_256_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_256_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8 * pMessage,
                                       unsigned U8 MessageLen,
                                       unsigned U8 * pSalt,
                                       const U8 SaltLen,
                                       unsigned * pSignature,
                                       CRYPTO_MEM_CONTEXT * pSignatureLen,
                                       * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.19 CRYPTO_RSASSA_PSS_SHA3_384_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_384_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.20 CRYPTO_RSASSA_PSS_SHA3_384_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_384_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8 * pMessage,
                                       unsigned U8 MessageLen,
                                       unsigned U8 * pSalt,
                                       const U8 SaltLen,
                                       unsigned * pSignature,
                                       CRYPTO_MEM_CONTEXT * pSignatureLen,
                                       * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.5.21 CRYPTO_RSASSA_PSS_SHA3_512_Sign()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_512_Sign(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                      const U8                           * pMessage,
                                      unsigned                         MessageLen,
                                      const U8                           * pSalt,
                                      unsigned                         SaltLen,
                                      U8                               * pSignature,
                                      unsigned                         SignatureLen,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to sign the message with.
pMessage	Pointer to message to sign.
MessageLen	Size of message to sign in bytes.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.5.22 CRYPTO_RSASSA_PSS_SHA3_512_Verify()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_512_Verify(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                       const U8 * pMessage,
                                       unsigned U8 MessageLen,
                                       unsigned U8 * pSalt,
                                       const U8 SaltLen,
                                       unsigned * pSignature,
                                       CRYPTO_MEM_CONTEXT * pSignatureLen,
                                       * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pMessage	Message to verify.
MessageLen	Size of message in bytes.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6 RSASSA-PSS digest sign and verify

The following table lists the RSASSA-PSS type-safe sigest sign and verify API functions.

Function	Description
Sign digest	
<code>CRYPTO_RSASSA_PSS_SHA1_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA224_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA256_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA384_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_224_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_256_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_224_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_256_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_384_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_512_SignDigest()</code>	Signs a message with a private key using the RSASSA-PSS-Sign algorithm.
Verify digest	
<code>CRYPTO_RSASSA_PSS_SHA1_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA224_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA256_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA384_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_224_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA512_256_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_224_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_256_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_384_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.
<code>CRYPTO_RSASSA_PSS_SHA3_512_VerifyDigest()</code>	Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

13.1.6.1 CRYPTO_RSASSA_PSS_SHA1_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA1_SignDigest(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                       const U8                      * pDigest,
                                       const U8                      * pSalt,
                                       unsigned                     SaltLen,
                                       U8                           * pSignature,
                                       unsigned                     SignatureLen,
                                       CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA1 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.2 CRYPTO_RSASSA_PSS_SHA1_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA1_VerifyDigest(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         const U8                      * pDigest,
                                         U8                           * pSalt,
                                         unsigned                     SaltLen,
                                         const U8                      * pSignature,
                                         unsigned                     SignatureLen,
                                         CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.3 CRYPTO_RSASSA_PSS_SHA224_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA224 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.4 CRYPTO_RSASSA_PSS_SHA224_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.5 CRYPTO_RSASSA_PSS_SHA256_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA256 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.6 CRYPTO_RSASSA_PSS_SHA256_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.7 CRYPTO_RSASSA_PSS_SHA384_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA384_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA384 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.8 CRYPTO_RSASSA_PSS_SHA384_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA384_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.9 CRYPTO_RSASSA_PSS_SHA512_224_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA512_224 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.10 CRYPTO_RSASSA_PSS_SHA512_224_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.11 CRYPTO_RSASSA_PSS_SHA512_256_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA512_256 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.12 CRYPTO_RSASSA_PSS_SHA512_256_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.13 CRYPTO_RSASSA_PSS_SHA512_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA512 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.14 CRYPTO_RSASSA_PSS_SHA512_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA512_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.15 CRYPTO_RSASSA_PSS_SHA3_224_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_224_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA3_224 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.16 CRYPTO_RSASSA_PSS_SHA3_224_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_224_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.17 CRYPTO_RSASSA_PSS_SHA3_256_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_256_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA3_256 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.18 CRYPTO_RSASSA_PSS_SHA3_256_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_256_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.19 CRYPTO_RSASSA_PSS_SHA3_384_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_384_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA3_384 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.20 CRYPTO_RSASSA_PSS_SHA3_384_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_384_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.6.21 CRYPTO_RSASSA_PSS_SHA3_512_SignDigest()

Description

Signs a message with a private key using the RSASSA-PSS-Sign algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_512_SignDigest
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     const U8                      * pDigest,
     const U8                      * pSalt,
     unsigned                     SaltLen,
     U8                           * pSignature,
     unsigned                     SignatureLen,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key to sign the message with.
pDigest	Pointer to SHA3_512 hash of the original message.
pSalt	Salt value to embed.
SaltLen	Size of salt in bytes.
pSignature	Pointer to object that receives the generated signature.
SignatureLen	Size of signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but signature failure (signature buffer too small).
- > 0 Nonzero indicates the number of bytes written to the the signature buffer that constitute the signature.

13.1.6.22 CRYPTO_RSASSA_PSS_SHA3_512_VerifyDigest()

Description

Verify a message using a public key and the RSASSA-PSS-Verify algorithm.

Prototype

```
int CRYPTO_RSASSA_PSS_SHA3_512_VerifyDigest
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     const U8                  * pDigest,
     U8                         * pSalt,
     unsigned                   SaltLen,
     const U8                  * pSignature,
     unsigned                   SignatureLen,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Public key used to verify the message.
pDigest	Hash of original message to be verified.
pSalt	Recovered salt. If pSalt is null, the salt is not recovered, but SaltLen must still be given.
SaltLen	Length of the original salt.
pSignature	Signature to verify.
SignatureLen	Size of the signature in bytes.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status indication.
- = 0 Processing complete but verification failure.
- > 0 Signature verified successfully.

13.1.7 Self-test API

The following table lists the RSA self-test API functions.

Function	Description
RSASSA sign and verify	
CRYPTO_RSASSA_PKCS1_Sign_CAVS_SelfTest()	Run CAVS RSA signing test vectors.
CRYPTO_RSASSA_PKCS1_Sign_EMCA_SelfTest()	Run RSA-PKCS1 test vectors from EMC.
CRYPTO_RSASSA_PKCS1_Verify_CAVS_SelfTest()	Run CAVS RSA signature verification vectors.
CRYPTO_RSASSA_PSS_Sign_CAVS_SelfTest()	Run CAVS RSA signing test vectors.
CRYPTO_RSASSA_PSS_Sign_EMCA_SelfTest()	Run RSA-PSS test vectors from EMC.
CRYPTO_RSASSA_PSS_Verify_CAVS_SelfTest()	Run CAVS RSA signature verification vectors.
RSA key generation	
CRYPTO_RSA_SHA1_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA224_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA256_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA384_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA512_224_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA512_256_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
CRYPTO_RSA_SHA512_KeyGen_CAVS_SelfTest()	Run RSA key generation KATs from CAVS.
Extended self tests	
CRYPTO_RSA_SEGGER_SelfTest()	Run RSA self tests from SEGGER.

13.1.7.1 CRYPTO_RSASSA_PKCS1_Sign_CAVS_SelfTest()

Description

Run CAVS RSA signing test vectors.

Prototype

```
void CRYPTO_RSASSA_PKCS1_Sign_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.2 CRYPTO_RSASSA_PKCS1_Sign_EMCSelfTest()

Description

Run RSA-PKCS1 test vectors from EMC.

Prototype

```
void CRYPTO_RSASSA_PKCS1_Sign_EMCSelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.3 CRYPTO_RSASSA_PKCS1_Verify_CAVS_SelfTest()

Description

Run CAVS RSA signature verification vectors.

Prototype

```
void CRYPTO_RSASSA_PKCS1_Verify_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.4 CRYPTO_RSASSA_PSS_Sign_CAVS_SelfTest()

Description

Run CAVS RSA signing test vectors.

Prototype

```
void CRYPTO_RSASSA_PSS_Sign_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.5 CRYPTO_RSASSA_PSS_Sign_EMCSelfTest()

Description

Run RSA-PSS test vectors from EMC.

Prototype

```
void CRYPTO_RSASSA_PSS_Sign_EMCSelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.6 CRYPTO_RSASSA_PSS_Verify_CAVS_SelfTest()

Description

Run CAVS RSA signature verification vectors.

Prototype

```
void CRYPTO_RSASSA_PSS_Verify_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.7 CRYPTO_RSA_SHA1_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA1_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.8 CRYPTO_RSA_SHA224_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA224_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.9 CRYPTO_RSA_SHA256_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA256_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.10 CRYPTO_RSA_SHA384_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA384_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.11 CRYPTO_RSA_SHA512_224_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA512_224_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.12 CRYPTO_RSA_SHA512_256_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA512_256_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.13 CRYPTO_RSA_SHA512_KeyGen_CAVS_SelfTest()

Description

Run RSA key generation KATs from CAVS.

Prototype

```
void CRYPTO_RSA_SHA512_KeyGen_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.1.7.14 CRYPTO_RSA SEGGER SelfTest()

Description

Run RSA self tests from SEGGER.

Prototype

```
void CRYPTO_RSA SEGGER SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

13.2 DSA

13.2.1 Management

Function	Description
<code>CRYPTO_DSA_InitDomainParams()</code>	Initialize domain parameters for use.
<code>CRYPTO_DSA_InitPublicKey()</code>	Initialize public key for use.
<code>CRYPTO_DSA_InitPrivateKey()</code>	Initialize private key for use.
<code>CRYPTO_DSA_InitSignature()</code>	Initialize signature for use.
<code>CRYPTO_DSA_KillDomainParams()</code>	Zero all data relating to the DSA domain parameters and reclaim storage.
<code>CRYPTO_DSA_KillPublicKey()</code>	Zero all data relating to the DSA public key and reclaim storage.
<code>CRYPTO_DSA_KillPrivateKey()</code>	Zero all data relating to the DSA private key and reclaim storage.
<code>CRYPTO_DSA_KillSignature()</code>	Zero all data relating to the DSA signature and reclaim storage.

13.2.1.1 CRYPTO_DSA_InitDomainParams()

Description

Initialize domain parameters for use.

Prototype

```
void CRYPTO_DSA_InitDomainParams(CRYPTO_DSA_DOMAIN_PARAMS * pSelf,  
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Domain parameters to initialize.
pMem	Allocator to use for temporary storage.

13.2.1.2 CRYPTO_DSA_InitPrivateKey()

Description

Initialize private key for use.

Prototype

```
void CRYPTO_DSA_InitPrivateKey(CRYPTO_DSA_PRIVATE_KEY * pSelf,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to initialize.
pMem	Allocator to use for temporary storage.

13.2.1.3 CRYPTO_DSA_InitPublicKey()

Description

Initialize public key for use.

Prototype

```
void CRYPTO_DSA_InitPublicKey(CRYPTO_DSA_PUBLIC_KEY * pSelf,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to initialize.
pMem	Allocator to use for temporary storage.

13.2.1.4 CRYPTO_DSA_InitSignature()

Description

Initialize signature for use.

Prototype

```
void CRYPTO_DSA_InitSignature(CRYPTO_DSA_SIGNATURE * pSelf,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Signature to initialize.
pMem	Allocator to use for temporary storage.

13.2.1.5 CRYPTO_DSA_KillDomainParams()

Description

Zero all data relating to the DSA domain parameters and reclaim storage.

Prototype

```
void CRYPTO_DSA_KillDomainParams(CRYPTO_DSA_DOMAIN_PARAMS * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Domain parameters to clear.

13.2.1.6 CRYPTO_DSA_KillPrivateKey()

Description

Zero all data relating to the DSA private key and reclaim storage.

Prototype

```
void CRYPTO_DSA_KillPrivateKey(CRYPTO_DSA_PRIVATE_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Private key to clear.

13.2.1.7 CRYPTO_DSA_KillPublicKey()

Description

Zero all data relating to the DSA public key and reclaim storage.

Prototype

```
void CRYPTO_DSA_KillPublicKey(CRYPTO_DSA_PUBLIC_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Public key to clear.

13.2.1.8 CRYPTO_DSA_KillSignature()

Description

Zero all data relating to the DSA signature and reclaim storage.

Prototype

```
void CRYPTO_DSA_KillSignature(CRYPTO_DSA_SIGNATURE * pSelf);
```

Parameters

Parameter	Description
pSelf	Signature to clear.

13.2.2 Key generation

Function	Description
<code>CRYPTO_DSA_GenDomainParas()</code>	Generate DSA domain parameters for keys of length L bits according to FIPS 186-1.
<code>CRYPTO_DSA_GenKeys()</code>	Generate user DSA public and private keys given for a set of DSA domain parameters.
<code>CRYPTO_DSA_FIPS186_ValidateParas()</code>	Validate that the prime sizes L and N for P and Q are acceptable by the FIPS 186-4 standard.
<code>CRYPTO_DSA_FIPS186_GenDomainParas()</code>	Generate DSA domain parameters for keys of given lengths according to FIPS 186-4 by choosing a random starting seed.

13.2.2.1 CRYPTO_DSA_FIPS186_GenDomainParas()

Description

Generate DSA domain parameters for keys of given lengths according to FIPS 186-4 by choosing a random starting seed.

Prototype

```
int CRYPTO_DSA_FIPS186_GenDomainParas
    (
        CRYPTO_DSA_DOMAIN_PARAMS * pParas,
        unsigned L,
        unsigned N,
        const CRYPTO_FIPS186_PRIMEGEN_API * pPrimeAPI,
        CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParas	Generated DSA domain parameters.
L	Strength in bits, of the prime Q.
N	Strength in bits, of the prime P.
pPrimeAPI	Pointer to prime generation API.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Failed to generate domain parameters from the seed.
- ≥ 0 Successful generation.

13.2.2.2 CRYPTO_DSA_FIPS186_ValidateParas()

Description

Validate that the prime sizes [L](#) and [N](#) for P and Q are acceptable by the FIPS 186-4 standard.

Prototype

```
int CRYPTO_DSA_FIPS186_ValidateParas(unsigned L,  
                                     unsigned N);
```

Parameters

Parameter	Description
L	Strength in bits of the prime Q.
N	Strength in bits of the prime P.

Return value

- = 0 Parameters are not acceptable.
- ≠ 0 Parameters are valid.

13.2.2.3 CRYPTO_DSA_GenDomainParas()

Description

Generate DSA domain parameters for keys of length `L` bits according to FIPS 186-1. An optional feedback function can be supplied which charts the progress of the DSA generation algorithm.

Prototype

```
int CRYPTO_DSA_GenDomainParas(CRYPTO_DSA_DOMAIN_PARAMS * pParas,  
                               unsigned L,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
<code>pParas</code>	Generated DSA domain parameters.
<code>L</code>	Strength in bits, $512 \leq L \leq 1024$ and a multiple of 64.
<code>pMem</code>	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

Additional information

FIPS 186-4 DSA domain generation with proven primes is offered by `CRYPTO_DSA_FIPS186_GenerateDomainParams()`.

13.2.2.4 CRYPTO_DSA_GenKeys()

Description

Generate user DSA public and private keys given for a set of DSA domain parameters.

Prototype

```
int CRYPTO_DSA_GenKeys( const CRYPTO_DSA_DOMAIN_PARAMS * pParams,  
                        CRYPTO_DSA_PRIVATE_KEY    * pPrivateKey,  
                        CRYPTO_DSA_PUBLIC_KEY     * pPublicKey,  
                        CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters used to generate user keys.
pPrivateKey	Generated user private key.
pPublicKey	Generated user public key.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

13.2.3 Message sign and verify

Function	Description
Signing	
<code>CRYPTO_DSA_SignDigest()</code>	Sign a message digest.
<code>CRYPTO_DSA_SHA1_Sign()</code>	Sign message using DSA-SHA-1.
<code>CRYPTO_DSA_SHA224_Sign()</code>	Sign message using DSA-SHA-224.
<code>CRYPTO_DSA_SHA256_Sign()</code>	Sign message using DSA-SHA-256.
<code>CRYPTO_DSA_SHA384_Sign()</code>	Sign message using DSA-SHA-384.
<code>CRYPTO_DSA_SHA512_Sign()</code>	Sign message using DSA-SHA-512.
<code>CRYPTO_DSA_SHA512_224_Sign()</code>	Sign message using DSA-SHA-512/224.
<code>CRYPTO_DSA_SHA512_256_Sign()</code>	Sign message using DSA-SHA-512/256.
<code>CRYPTO_DSA_SHA3_224_Sign()</code>	Sign message using DSA-SHA3-224.
<code>CRYPTO_DSA_SHA3_256_Sign()</code>	Sign message using DSA-SHA3-256.
<code>CRYPTO_DSA_SHA3_384_Sign()</code>	Sign message using DSA-SHA3-384.
<code>CRYPTO_DSA_SHA3_512_Sign()</code>	Sign message using DSA-SHA3-512.
<code>CRYPTO_DSA_SignDigestWithK()</code>	Sign a message using the given DSA domain parameters and private key, and a predetermined value of K, generating a signature.
Deterministic signing	
<code>CRYPTO_DSA_RFC6979_SHA1_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA224_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA256_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA384_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA512_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA512_224_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA512_256_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_DSA_RFC6979_SHA3_224_Sign()</code>	Sign a message using the given DSA domain parameters and private key,

Function	Description
	generating a signature using Deterministic DSA to choose K.
CRYPTO_DSA_RFC6979_SHA3_256_Sign()	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
CRYPTO_DSA_RFC6979_SHA3_384_Sign()	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
CRYPTO_DSA_RFC6979_SHA3_512_Sign()	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
Verification	
CRYPTO_DSA_IsValidSignature()	Return whether the signature (R, S) is a valid signature.
CRYPTO_DSA_VerifyDigest()	Verify the signature of a message digest using the given DSA domain parameters and public key.
CRYPTO_DSA_SHA1_Verify()	Verify message using DSA-SHA-1.
CRYPTO_DSA_SHA224_Verify()	Verify message using DSA-SHA-224.
CRYPTO_DSA_SHA256_Verify()	Verify message using DSA-SHA-256.
CRYPTO_DSA_SHA384_Verify()	Verify message using DSA-SHA-384.
CRYPTO_DSA_SHA512_Verify()	Verify message using DSA-SHA-512.
CRYPTO_DSA_SHA512_224_Verify()	Verify message using DSA-SHA-512/224.
CRYPTO_DSA_SHA512_256_Verify()	Verify message using DSA-SHA-512/256.
CRYPTO_DSA_SHA3_224_Verify()	Verify message using DSA-SHA3-224.
CRYPTO_DSA_SHA3_256_Verify()	Verify message using DSA-SHA3-256.
CRYPTO_DSA_SHA3_384_Verify()	Verify message using DSA-SHA3-384.
CRYPTO_DSA_SHA3_512_Verify()	Verify message using DSA-SHA3-512.

13.2.3.1 CRYPTO_DSA_IsValidSignature()

Description

Return whether the signature (R, S) is a valid signature. Signatures are considered invalid if either R or S components are zero, in which case a new K should be computed and the signature tried again.

Prototype

```
int CRYPTO_DSA_IsValidSignature(const CRYPTO_DSA_SIGNATURE * pSelf);
```

Parameters

Parameter	Description
pSelf	Signature to test.

Return value

Boolean indicating a valid signature.

13.2.3.2 CRYPTO_DSA_RFC6979_SHA1_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA1_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                  const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                  const U8 * pMessage,
                                  unsigned MessageLen,
                                  CRYPTO_DSA_SIGNATURE * pSignature,
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.3 CRYPTO_DSA_RFC6979_SHA224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                     const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                     const U8                           * pMessage,
                                     unsigned                         MessageLen,
                                     CRYPTO_DSA_SIGNATURE      * pSignature,
                                     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.4 CRYPTO_DSA_RFC6979_SHA256_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                     const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                     const U8 * pMessage,
                                     unsigned MessageLen,
                                     CRYPTO_DSA_SIGNATURE * pSignature,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.5 CRYPTO_DSA_RFC6979_SHA384_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA384_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                     const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                     const U8                           * pMessage,
                                     unsigned                         MessageLen,
                                     CRYPTO_DSA_SIGNATURE      * pSignature,
                                     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.6 CRYPTO_DSA_RFC6979_SHA512_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA512_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                     const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                     const U8                           * pMessage,
                                     unsigned                         MessageLen,
                                     CRYPTO_DSA_SIGNATURE      * pSignature,
                                     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.7 CRYPTO_DSA_RFC6979_SHA512_224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA512_224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                         const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         CRYPTO_DSA_SIGNATURE * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.8 CRYPTO_DSA_RFC6979_SHA512_256_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA512_256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                         const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         CRYPTO_DSA_SIGNATURE * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.9 CRYPTO_DSA_RFC6979_SHA3_224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA3_224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                       const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                       const U8 * pMessage,
                                       unsigned MessageLen,
                                       CRYPTO_DSA_SIGNATURE * pSignature,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.10 CRYPTO_DSA_RFC6979_SHA3_256_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA3_256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                       const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                       const U8 * pMessage,
                                       unsigned MessageLen,
                                       CRYPTO_DSA_SIGNATURE * pSignature,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.11 CRYPTO_DSA_RFC6979_SHA3_384_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA3_384_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                       const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                       const U8 * pMessage,
                                       unsigned MessageLen,
                                       CRYPTO_DSA_SIGNATURE * pSignature,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.12 CRYPTO_DSA_RFC6979_SHA3_512_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_DSA_RFC6979_SHA3_512_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                       const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                       const U8 * pMessage,
                                       unsigned MessageLen,
                                       CRYPTO_DSA_SIGNATURE * pSignature,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.2.3.13 CRYPTO_DSA_SHA1_Sign()

Description

Sign message using DSA-SHA-1.

Prototype

```
int CRYPTO_DSA_SHA1_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                         const CRYPTO_DSA_PRIVATE_KEY * pKey,
                         const U8 * pMessage,
                         unsigned MessageLen,
                         CRYPTO_DSA_SIGNATURE * pSignature,
                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-1 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.14 CRYPTO_DSA_SHA224_Sign()

Description

Sign message using DSA-SHA-224.

Prototype

```
int CRYPTO_DSA_SHA224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY * pKey,
                           const U8 unsigned
                           CRYPTO_DSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-224 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.15 CRYPTO_DSA_SHA256_Sign()

Description

Sign message using DSA-SHA-256.

Prototype

```
int CRYPTO_DSA_SHA256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY * pKey,
                           const U8 unsigned
                           CRYPTO_DSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-256 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.16 CRYPTO_DSA_SHA384_Sign()

Description

Sign message using DSA-SHA-384.

Prototype

```
int CRYPTO_DSA_SHA384_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY * pKey,
                           const U8 unsigned
                           CRYPTO_DSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-384 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.17 CRYPTO_DSA_SHA512_Sign()

Description

Sign message using DSA-SHA-512.

Prototype

```
int CRYPTO_DSA_SHA512_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY * pKey,
                           const U8 unsigned
                           CRYPTO_DSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-512 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.18 CRYPTO_DSA_SHA512_224_Sign()

Description

Sign message using DSA-SHA-512/224.

Prototype

```
int CRYPTO_DSA_SHA512_224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                const U8 * pMessage,
                                unsigned MessageLen,
                                CRYPTO_DSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-512/224 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.19 CRYPTO_DSA_SHA512_256_Sign()

Description

Sign message using DSA-SHA-512/256.

Prototype

```
int CRYPTO_DSA_SHA512_256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                const U8 * pMessage,
                                unsigned MessageLen,
                                CRYPTO_DSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA-512/256 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.20 CRYPTO_DSA_SHA3_224_Sign()

Description

Sign message using DSA-SHA3-224.

Prototype

```
int CRYPTO_DSA_SHA3_224_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                               const CRYPTO_DSA_PRIVATE_KEY * pKey,
                               const U8 * pMessage,
                               unsigned MessageLen,
                               CRYPTO_DSA_SIGNATURE * pSignature,
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA3-224 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.21 CRYPTO_DSA_SHA3_256_Sign()

Description

Sign message using DSA-SHA3-256.

Prototype

```
int CRYPTO_DSA_SHA3_256_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                               const CRYPTO_DSA_PRIVATE_KEY * pKey,
                               const U8 * pMessage,
                               unsigned MessageLen,
                               CRYPTO_DSA_SIGNATURE * pSignature,
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA3-256 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.22 CRYPTO_DSA_SHA3_384_Sign()

Description

Sign message using DSA-SHA3-384.

Prototype

```
int CRYPTO_DSA_SHA3_384_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY * pKey,
                           const U8 * pMessage,
                           unsigned MessageLen,
                           CRYPTO_DSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA3-384 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.23 CRYPTO_DSA_SHA3_512_Sign()

Description

Sign message using DSA-SHA3-512.

Prototype

```
int CRYPTO_DSA_SHA3_512_Sign(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                               const CRYPTO_DSA_PRIVATE_KEY * pKey,
                               const U8 * pMessage,
                               unsigned MessageLen,
                               CRYPTO_DSA_SIGNATURE * pSignature,
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Computes the SHA3-512 digest over the message and signs the message given the DSA domain parameters and private key to sign with.

13.2.3.24 CRYPTO_DSA_SignDigest()

Description

Sign a message digest. This uses the given DSA domain parameters and private key to sign the digest and generate a signature.

Prototype

```
int CRYPTO_DSA_SignDigest(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PRIVATE_KEY    * pKey,
                           const U8                      * pDigest,
                           unsigned                     DigestLen,
                           CRYPTO_DSA_SIGNATURE        * pSignature,
                           CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	User's DSA private key for signing.
pDigest	Pointer to digest octet string.
DigestLen	Octet length of the digest octet string.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

< 0	Processing error.
≥ 0	Success.

13.2.3.25 CRYPTO_DSA_SignDigestWithK()

Description

Sign a message using the given DSA domain parameters and private key, and a predetermined value of K, generating a signature. This is primarily of use when running the DSA test vectors or when you compute K deterministically from the message, such as in [RFC6979].

Prototype

```
int CRYPTO_DSA_SignDigestWithK(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PRIVATE_KEY * pKey,
                                const U8 * pDigest,
                                unsigned DigestLen,
                                const CRYPTO_MPI * pK,
                                CRYPTO_DSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	User's private key for signing.
pDigest	Pointer to digest octet string.
DigestLen	Octet length of the digest octet string.
pK	Secret K value to use when signing.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Processing complete but no valid signature generated — S, R, or both are zero and the caller should generate a new K.
- > 0 Success — the computed signature is valid for the given K with both S and R nonzero.

13.2.3.26 CRYPTO_DSA_VerifyDigest()

Description

Verify the signature of a message digest using the given DSA domain parameters and public key.

Prototype

```
int CRYPTO_DSA_VerifyDigest(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pDigest,
                           unsigned                        DigestLen,
                           const CRYPTO_DSA_SIGNATURE     * pSignature,
                           CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pParams	DSA domain parameters.
pKey	User's public key.
pDigest	Pointer to digest octet string.
DigestLen	Octet length of the digest octet string.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.27 CRYPTO_DSA_SHA1_Verify()

Description

Verify message using DSA-SHA-1.

Prototype

```
int CRYPTO_DSA_SHA1_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pMessage,
                           unsigned                        MessageLen,
                           const CRYPTO_DSA_SIGNATURE      * pSignature,
                           CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.28 CRYPTO_DSA_SHA224_Verify()

Description

Verify message using DSA-SHA-224.

Prototype

```
int CRYPTO_DSA_SHA224_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pMessage,
                           unsigned                         MessageLen,
                           const CRYPTO_DSA_SIGNATURE      * pSignature,
                           CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.29 CRYPTO_DSA_SHA256_Verify()

Description

Verify message using DSA-SHA-256.

Prototype

```
int CRYPTO_DSA_SHA256_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pMessage,
                           unsigned                         MessageLen,
                           const CRYPTO_DSA_SIGNATURE      * pSignature,
                           CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.30 CRYPTO_DSA_SHA384_Verify()

Description

Verify message using DSA-SHA-384.

Prototype

```
int CRYPTO_DSA_SHA384_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pMessage,
                           unsigned                         MessageLen,
                           const CRYPTO_DSA_SIGNATURE      * pSignature,
                           CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.31 CRYPTO_DSA_SHA512_Verify()

Description

Verify message using DSA-SHA-512.

Prototype

```
int CRYPTO_DSA_SHA512_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                           const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                           const U8                         * pMessage,
                           unsigned                         MessageLen,
                           const CRYPTO_DSA_SIGNATURE      * pSignature,
                           CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.32 CRYPTO_DSA_SHA512_224_Verify()

Description

Verify message using DSA-SHA-512/224.

Prototype

```
int CRYPTO_DSA_SHA512_224_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                  const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                  const U8                         * pMessage,
                                  unsigned                        MessageLen,
                                  const CRYPTO_DSA_SIGNATURE     * pSignature,
                                  CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.33 CRYPTO_DSA_SHA512_256_Verify()

Description

Verify message using DSA-SHA-512/256.

Prototype

```
int CRYPTO_DSA_SHA512_256_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                  const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                  const U8                         * pMessage,
                                  unsigned                         MessageLen,
                                  const CRYPTO_DSA_SIGNATURE      * pSignature,
                                  CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.34 CRYPTO_DSA_SHA3_224_Verify()

Description

Verify message using DSA-SHA3-224.

Prototype

```
int CRYPTO_DSA_SHA3_224_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                const U8                         * pMessage,
                                unsigned                         MessageLen,
                                const CRYPTO_DSA_SIGNATURE      * pSignature,
                                CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.35 CRYPTO_DSA_SHA3_256_Verify()

Description

Verify message using DSA-SHA3-256.

Prototype

```
int CRYPTO_DSA_SHA3_256_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                const U8                         * pMessage,
                                unsigned                         MessageLen,
                                const CRYPTO_DSA_SIGNATURE      * pSignature,
                                CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.36 CRYPTO_DSA_SHA3_384_Verify()

Description

Verify message using DSA-SHA3-384.

Prototype

```
int CRYPTO_DSA_SHA3_384_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                const U8                         * pMessage,
                                unsigned                         MessageLen,
                                const CRYPTO_DSA_SIGNATURE      * pSignature,
                                CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.3.37 CRYPTO_DSA_SHA3_512_Verify()

Description

Verify message using DSA-SHA3-512.

Prototype

```
int CRYPTO_DSA_SHA3_512_Verify(const CRYPTO_DSA_DOMAIN_PARAMS * pParams,
                                const CRYPTO_DSA_PUBLIC_KEY      * pKey,
                                const U8                         * pMessage,
                                unsigned                         MessageLen,
                                const CRYPTO_DSA_SIGNATURE      * pSignature,
                                CRYPTO_MEM_CONTEXT             * pMem);
```

Parameters

Parameter	Description
pParams	Pointer to group parameters.
pKey	Pointer to user's private key for signing.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Pointer to memory allocator context for temporary storage.

Return value

- < 0 Processing error verifying digest. 0 - Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.2.4 Self-test API

The following table lists the DSA self-test API functions.

Function	Description
<code>CRYPTO_DSA SEGGER_SelfTest()</code>	Run DSA self tests from SEGGER.
<code>CRYPTO_DSA SHA1_CAVS_SelfTest()</code>	Run CAVS DSA signature verification vectors.
<code>CRYPTO_DSA SHA224_CAVS_SelfTest()</code>	Run CAVS DSA signature verification vectors.
<code>CRYPTO_DSA SHA256_CAVS_SelfTest()</code>	Run CAVS DSA signature verification vectors.
<code>CRYPTO_DSA SHA384_CAVS_SelfTest()</code>	Run CAVS DSA signature verification vectors.
<code>CRYPTO_DSA SHA512_CAVS_SelfTest()</code>	Run CAVS DSA signature verification vectors.

13.2.4.1 CRYPTO_DSA SEGGER_SelfTest()

Description

Run DSA self tests from SEGGER.

Prototype

```
void CRYPTO_DSA SEGGER_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

13.2.4.2 CRYPTO_DSA_SHA1_CAVS_SelfTest()

Description

Run CAVS DSA signature verification vectors.

Prototype

```
void CRYPTO_DSA_SHA1_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.2.4.3 CRYPTO_DSA_SHA224_CAVS_SelfTest()

Description

Run CAVS DSA signature verification vectors.

Prototype

```
void CRYPTO_DSA_SHA224_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.2.4.4 CRYPTO_DSA_SHA256_CAVS_SelfTest()

Description

Run CAVS DSA signature verification vectors.

Prototype

```
void CRYPTO_DSA_SHA256_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.2.4.5 CRYPTO_DSA_SHA384_CAVS_SelfTest()

Description

Run CAVS DSA signature verification vectors.

Prototype

```
void CRYPTO_DSA_SHA384_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.2.4.6 CRYPTO_DSA_SHA512_CAVS_SelfTest()

Description

Run CAVS DSA signature verification vectors.

Prototype

```
void CRYPTO_DSA_SHA512_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.3 ECDSA

13.3.1 Introduction

A ECDSA key pair consists of a private key (which can be used to create signatures) and a public key (which can be used to verify signatures).

ECDSA keys

An ECDSA private key is just an octet string, but has to be provided as object of type CRYPTO_ECDSA_PRIVATE_KEY to the API functions. It is stored as multi precision integer in the CRYPTO_ECDSA_PRIVATE_KEY object.

The ECDSA public key has to be provided as object of type CRYPTO_ECDSA_PUBLIC_KEY to the API functions. It consists of two octet string (coordinates x and y of an elliptic curve point) and are stored as multi precision integers in the CRYPTO_ECDSA_PUBLIC_KEY object.

In most cases the keys are not available as object of this types in the application. Therefore the application has to load the keys into a CRYPTO_ECDSA_PUBLIC_KEY or CRYPTO_ECDSA_PRIVATE_KEY object respectively before it can be used by cryptographic functions. This can be done using the multi precision integer function described in *Format conversion* on page 2608. Depending of the format the private key is available, an appropriate conversion function can be chosen. See the examples below.

ECDSA signatures

An ECDSA signature consists of two octet strings (called r and s) and is stored as multi precision integers in an object of type CRYPTO_ECDSA_SIGNATURE to be used by the API functions. In most cases the signature needs to be converted by the application. This can be done using the multi precision integer function described in *Format conversion* on page 2608. See the examples below.

Example (ECDSA signing)

```
const U8 Key[] = { 0x66, 0xee, 0xc9, 0xa, 0x05, 0x15, 0xc42, ..., 0x87 };

CRYPTO_ECDSA_PRIVATE_KEY PrivateKey;
CRYPTO_ECDSA_SIGNATURE Signature;
U8 SigR[24];
U8 SigS[24];
// Load private key
// CRYPTO_ECDSA_InitPrivateKey(&PrivateKey, &MemContext);
if (CRYPTO_MPI_LoadBytes(&PrivateKey.X, Key, sizeof(Key)) < 0) {
    // handle error
}
// Sign message
// CRYPTO_ECDSA_InitSignature(&Signature, &MemContext);
r = CRYPTO_ECDSA_SHA1_Sign(&CRYPTO_EC_Curve_P192, &PrivateKey,
                           pMessage, MessageLen, &Signature, &MemContext);
if (r < 0) {
    // handle error
}
// Store signature to SigR, SigS
// CRYPTO_MPI_StoreBytes(&Signature.R, SigR, sizeof(SigR));
CRYPTO_MPI_StoreBytes(&Signature.S, SigS, sizeof(SigS));
```

For an explanation of the memory context MemContext refer to *Dynamic memory usage* on page 22.

Example (ECDSA verify)

```
const U8 KeyX[] = { 0x18, 0xcc, 0xee, 0xed, 0xc2, 0xdf, 0x74, ..., 0x2a };
const U8 KeyY[] = { 0x86, 0x5f, 0xe0, 0xa9, 0x25, 0x34, 0xa4, ..., 0xd9 };

CRYPTO_ECDSA_PUBLIC_KEY PublicKey;
CRYPTO_ECDSA_SIGNATURE Signature;
U8 SigR[24];
U8 SigS[24];
// Load public key
// CRYPTO_ECDSA_InitPublicKey(&PublicKey, &MemContext);
if (CRYPTO_MPI_LoadBytes(&PublicKey.Y.X, KeyX, sizeof(KeyX)) < 0 ||
    CRYPTO_MPI_LoadBytes(&PublicKey.Y.Y, KeyY, sizeof(KeyY)) < 0) {
    // handle error
}
// Load signature from SigR, SigS
// CRYPTO_ECDSA_InitSignature(&Signature, &MemContext);
if (CRYPTO_MPI_LoadBytes(&Signature.R, SigR, sizeof(SigR)) < 0 ||
    CRYPTO_MPI_LoadBytes(&Signature.S, SigS, sizeof(SigS)) < 0) {
    // handle error
}
// Verify signature
// r = CRYPTO_ECDSA_SHA1_Verify(&CRYPTO_EC_Curve_P192, &PublicKey,
//                               pMessage, MessageLen, &Signature, &MemContext);
if (r < 0) {
    // handle error
}
if (r == 0) {
    // bad signature
} else {
    // verification successful
}
```

For an explanation of the memory context `MemContext` refer to *Dynamic memory usage* on page 22.

13.3.2 Management

The following table lists the ECDSA key management API.

Function	Description
<code>CRYPTO_ECDSA_InitPublicKey()</code>	Initialize public key for use.
<code>CRYPTO_ECDSA_InitPrivateKey()</code>	Initialize private key for use.
<code>CRYPTO_ECDSA_InitSignature()</code>	Initialize signature for use.
<code>CRYPTO_ECDSA_KillPublicKey()</code>	Zero all data relating to the ECDSA public key and reclaim storage.
<code>CRYPTO_ECDSA_KillPrivateKey()</code>	Zero all data relating to the ECDSA private key and reclaim storage.
<code>CRYPTO_ECDSA_KillSignature()</code>	Zero all data relating to the DSA signature and reclaim storage.

13.3.2.1 CRYPTO_ECDSA_InitPrivateKey()

Description

Initialize private key for use.

Prototype

```
void CRYPTO_ECDSA_InitPrivateKey(CRYPTO_ECDSA_PRIVATE_KEY * pSelf,  
                                 CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Private key to initialize.
pMem	Allocator to use for temporary storage.

13.3.2.2 CRYPTO_ECDSA_InitPublicKey()

Description

Initialize public key for use.

Prototype

```
void CRYPTO_ECDSA_InitPublicKey(CRYPTO_ECDSA_PUBLIC_KEY * pSelf,  
                                CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to initialize.
pMem	Allocator to use for temporary storage.

13.3.2.3 CRYPTO_ECDSA_InitSignature()

Description

Initialize signature for use.

Prototype

```
void CRYPTO_ECDSA_InitSignature(CRYPTO_ECDSA_SIGNATURE * pSelf,  
                                CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Signature to initialize.
pMem	Allocator to use for temporary storage.

13.3.2.4 CRYPTO_ECDSA_KillPrivateKey()

Description

Zero all data relating to the ECDSA private key and reclaim storage.

Prototype

```
void CRYPTO_ECDSA_KillPrivateKey(CRYPTO_ECDSA_PRIVATE_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Private key to clear.

13.3.2.5 CRYPTO_ECDSA_KillPublicKey()

Description

Zero all data relating to the ECDSA public key and reclaim storage.

Prototype

```
void CRYPTO_ECDSA_KillPublicKey(CRYPTO_ECDSA_PUBLIC_KEY * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Public key to clear.

13.3.2.6 CRYPTO_ECDSA_KillSignature()

Description

Zero all data relating to the DSA signature and reclaim storage.

Prototype

```
void CRYPTO_ECDSA_KillSignature(CRYPTO_ECDSA_SIGNATURE * pSelf);
```

Parameters

Parameter	Description
pSelf	Signature to clear.

13.3.3 Key generation

The following table lists the ECDSA key generation API.

Function	Description
<code>CRYPTO_ECDSA_GenKeys()</code>	Generate user ECDSA public and private keys given for a set of ECDSA domain parameters.

13.3.3.1 CRYPTO_ECDSA_GenKeys()

Description

Generate user ECDSA public and private keys given for a set of ECDSA domain parameters.

Prototype

```
int CRYPTO_ECDSA_GenKeys( const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_ECDSA_PRIVATE_KEY * pPrivateKey,
                           CRYPTO_ECDSA_PUBLIC_KEY * pPublicKey,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to curve.
pPrivateKey	Generated user private key.
pPublicKey	Generated user public key.
pMem	Pointer to temporary storage allocator context.

Return value

< 0 Error status indication.
 ≥ 0 Success.

Additional information

As this function only supports NIST curves, there is no requirement to use cofactor multiplication to avoid small subgroup attacks as all NIST curves have cofactor 1.

13.3.4 Message sign and verify

The following table lists the ECDSA message sign and verify API.

Function	Description
Signing	
<code>CRYPTO_ECDSA_SHA1_Sign()</code>	Sign a message using the given ECDSA-SHA-1.
<code>CRYPTO_ECDSA_SHA224_Sign()</code>	Sign a message using the given ECDSA-SHA-224.
<code>CRYPTO_ECDSA_SHA256_Sign()</code>	Sign a message using the given ECDSA-SHA-256.
<code>CRYPTO_ECDSA_SHA384_Sign()</code>	Sign a message using the given ECDSA-SHA-384.
<code>CRYPTO_ECDSA_SHA512_Sign()</code>	Sign a message using the given ECDSA-SHA-512.
<code>CRYPTO_ECDSA_SHA512_224_Sign()</code>	Sign a message using the given ECDSA-SHA-512/224.
<code>CRYPTO_ECDSA_SHA512_256_Sign()</code>	Sign a message using the given ECDSA-SHA-512/256.
<code>CRYPTO_ECDSA_SHA3_224_Sign()</code>	Sign a message using the given ECDSA-SHA3-224.
<code>CRYPTO_ECDSA_SHA3_256_Sign()</code>	Sign a message using the given ECDSA-SHA3-256.
<code>CRYPTO_ECDSA_SHA3_384_Sign()</code>	Sign a message using the given ECDSA-SHA3-384.
<code>CRYPTO_ECDSA_SHA3_512_Sign()</code>	Sign a message using the given ECDSA-SHA3-512.
<code>CRYPTO_ECDSA_SignDigest()</code>	Sign a message using the given ECDSA domain parameters and private key, generating a signature.
<code>CRYPTO_ECDSA_SignDigestWithK()</code>	Sign a message using the given ECDSA domain parameters and private key, and a predetermined value of K, generating a signature.
Deterministic signing	
<code>CRYPTO_ECDSA_RFC6979_SHA1_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA224_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA256_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA384_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Function	Description
<code>CRYPTO_ECDsa_RFC6979_SHA512_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA512_224_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA512_256_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA3_224_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA3_256_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA3_384_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA3_512_Sign()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA1_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA224_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA256_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA384_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA512_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA512_224_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDsa_RFC6979_SHA512_256_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key,

Function	Description
	generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA3_224_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA3_256_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA3_384_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
<code>CRYPTO_ECDSA_RFC6979_SHA3_512_SignDigest()</code>	Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.
Verification	
<code>CRYPTO_ECDSA_VerifyDigest()</code>	Verify the signature of a message using the given ECDSA domain parameters and public key.
<code>CRYPTO_ECDSA_SHA1_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-1.
<code>CRYPTO_ECDSA_SHA224_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-224.
<code>CRYPTO_ECDSA_SHA256_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-256.
<code>CRYPTO_ECDSA_SHA384_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-384.
<code>CRYPTO_ECDSA_SHA512_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512.
<code>CRYPTO_ECDSA_SHA512_224_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512/224.
<code>CRYPTO_ECDSA_SHA512_256_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512/256.
<code>CRYPTO_ECDSA_SHA3_224_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-224.
<code>CRYPTO_ECDSA_SHA3_256_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-256.
<code>CRYPTO_ECDSA_SHA3_384_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-384.

Function	Description
<code>CRYPTO_ECDSA_SHA3_512_Verify()</code>	Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-512.
Deterministic K generation	
<code>CRYPTO_ECDSA_RFC6979_SHA1_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA224_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA256_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA384_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA512_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA512_224_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA512_256_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA3_224_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA3_256_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA3_384_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.
<code>CRYPTO_ECDSA_RFC6979_SHA3_512_GenK()</code>	Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

13.3.4.1 CRYPTO_ECDSA RFC6979 SHA1 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA1_GenK(const CRYPTO_EC_CURVE * pCurve,
                                     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                     const U8 * pDigest,
                                     CRYPTO_MPI * pK,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.2 CRYPTO_ECDSA RFC6979 SHA224 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA224 GenK( const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.3 CRYPTO_ECDSA RFC6979 SHA256 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA256 GenK( const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.4 CRYPTO_ECDSA RFC6979 SHA384 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA384 GenK( const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.5 CRYPTO_ECDSA RFC6979 SHA512 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA512 GenK(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.6 CRYPTO_ECDSA RFC6979 SHA512_224_GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA512_224_GenK(const CRYPTO_EC_CURVE * pCurve,
                                             const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                             const U8 * pDigest,
                                             CRYPTO_MPI * pK,
                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.7 CRYPTO_ECDSA RFC6979 SHA512_256_GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA512_256_GenK(const CRYPTO_EC_CURVE * pCurve,
                                             const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                             const U8 * pDigest,
                                             CRYPTO_MPI * pK,
                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.8 CRYPTO_ECDSA RFC6979 SHA3_224_GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA3_224_GenK(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.9 CRYPTO_ECDSA RFC6979 SHA3_256 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA3_256 GenK(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.10 CRYPTO_ECDSA RFC6979 SHA3_384 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_384_GenK(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.11 CRYPTO_ECDSA RFC6979 SHA3_512 GenK()

Description

Generates an acceptable K for signing a message using RFC6979 Deterministic DSA.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_512_GenK(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pDigest,
                                         CRYPTO_MPI * pK,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Private key to sign with.
pDigest	Subject digest to sign.
pK	Generated K.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.12 CRYPTO_ECDSA RFC6979 SHA1 Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA1_Sign(const CRYPTO_EC_CURVE * pCurve,
                                     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                     const U8 * pMessage,
                                     unsigned MessageLen,
                                     CRYPTO_ECDSA_SIGNATURE * pSignature,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.13 CRYPTO_ECDSA RFC6979 SHA224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA224_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE      * pSignature,
                                         CRYPTO_MEM_CONTEXT         * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.14 CRYPTO_ECDSA RFC6979 SHA256 Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA256_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE      * pSignature,
                                         CRYPTO_MEM_CONTEXT         * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.15 CRYPTO_ECDSA RFC6979 SHA384_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA384_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE      * pSignature,
                                         CRYPTO_MEM_CONTEXT         * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.16 CRYPTO_ECDSA RFC6979 SHA512 Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE      * pSignature,
                                         CRYPTO_MEM_CONTEXT         * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.17 CRYPTO_ECDSA RFC6979 SHA512_224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_224_Sign
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pMessage,
     unsigned                      MessageLen,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.18 CRYPTO_ECDSA RFC6979 SHA512_256_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_256_Sign
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pMessage,
     unsigned                      MessageLen,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.19 CRYPTO_ECDSA RFC6979 SHA3_224_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_224_Sign(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.20 CRYPTO_ECDSA RFC6979 SHA3_256_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_256_Sign(const CRYPTO_EC_CURVE * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8 * pMessage,
                                         unsigned MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE * pSignature,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.21 CRYPTO_ECDSA RFC6979 SHA3_384_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_384_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE       * pSignature,
                                         CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.22 CRYPTO_ECDSA RFC6979 SHA3_512_Sign()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_512_Sign(const CRYPTO_EC_CURVE          * pCurve,
                                         const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                         const U8                      * pMessage,
                                         unsigned                     MessageLen,
                                         CRYPTO_ECDSA_SIGNATURE       * pSignature,
                                         CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	Curve that points are embedded on.
pKey	Pointer to private key to sign with.
pMessage	Pointer to octet string containing the message to sign.
MessageLen	Octet length of message to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.23 CRYPTO_ECDSA RFC6979 SHA1 SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA1_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.24 CRYPTO_ECDSA RFC6979 SHA224_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA224_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.25 CRYPTO_ECDSA RFC6979 SHA256 SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA256_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.26 CRYPTO_ECDSA RFC6979 SHA384_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA384_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.27 CRYPTO_ECDSA RFC6979 SHA512 SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.28 CRYPTO_ECDSA RFC6979 SHA512_224_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_224_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.29 CRYPTO_ECDSA RFC6979 SHA512_256_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA512_256_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.30 CRYPTO_ECDSA RFC6979 SHA3_224_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA RFC6979 SHA3_224_SignDigest
    (const CRYPTO_EC_CURVE      * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE       * pSignature,
     CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.31 CRYPTO_ECDSA RFC6979 SHA3_256_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_256_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.32 CRYPTO_ECDSA RFC6979 SHA3_384_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_384_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.33 CRYPTO_ECDSA RFC6979 SHA3_512_SignDigest()

Description

Sign a message using the given DSA domain parameters and private key, generating a signature using Deterministic DSA to choose K.

Prototype

```
int CRYPTO_ECDSA_RFC6979_SHA3_512_SignDigest
    (const CRYPTO_EC_CURVE          * pCurve,
     const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
     const U8                      * pDigest,
     CRYPTO_ECDSA_SIGNATURE        * pSignature,
     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve group.
pKey	Pointer to private key.
pDigest	Pointer to digest to sign.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates the message cannot be signed, non-zero indicates that it can be signed.

13.3.4.34 CRYPTO_ECDSA_SHA1_Sign()

Description

Sign a message using the given ECDSA-SHA-1.

Prototype

```
int CRYPTO_ECDSA_SHA1_Sign(const CRYPTO_EC_CURVE * pParams,  
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,  
                           const U8 aMessage[],  
                           unsigned NumBytesMessage,  
                           CRYPTO_ECDSA_SIGNATURE * pSignature,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

13.3.4.35 CRYPTO_ECDSA_SHA1_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-1.

Prototype

```
int CRYPTO_ECDSA_SHA1_Verify(const CRYPTO_EC_CURVE          * pCurve,
                           const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                           const U8                      aMessage[],
                           unsigned                     MessageLen,
                           const CRYPTO_ECDSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.36 CRYPTO_ECDSA_SHA224_Sign()

Description

Sign a message using the given ECDSA-SHA-224.

Prototype

```
int CRYPTO_ECDSA_SHA224_Sign(const CRYPTO_EC_CURVE          * pParams,
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                           const U8                      aMessage[],
                           unsigned                     NumBytesMessage,
                           CRYPTO_ECDSA_SIGNATURE        * pSignature,
                           CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.37 CRYPTO_ECDSA_SHA224_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-224.

Prototype

```
int CRYPTO_ECDSA_SHA224_Verify(const CRYPTO_EC_CURVE * pCurve,
                                const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                const U8 aMessage[], unsigned aMessageLen,
                                const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.38 CRYPTO_ECDSA_SHA256_Sign()

Description

Sign a message using the given ECDSA-SHA-256.

Prototype

```
int CRYPTO_ECDSA_SHA256_Sign(const CRYPTO_EC_CURVE * pParams,
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                           const U8 aMessage[], NumBytesMessage,
                           unsigned CRYPTO_ECDSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.39 CRYPTO_ECDSA_SHA256_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-256.

Prototype

```
int CRYPTO_ECDSA_SHA256_Verify(const CRYPTO_EC_CURVE * pCurve,  
                                const CRYPTO_ECDSA_PUBLIC_KEY * pKey,  
                                const U8 aMessage[],  
                                unsigned MessageLen,  
                                const CRYPTO_ECDSA_SIGNATURE * pSignature,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.40 CRYPTO_ECDSA_SHA384_Sign()

Description

Sign a message using the given ECDSA-SHA-384.

Prototype

```
int CRYPTO_ECDSA_SHA384_Sign(const CRYPTO_EC_CURVE          * pParams,  
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,  
                           const U8                      aMessage[],  
                           unsigned                     NumBytesMessage,  
                           CRYPTO_ECDSA_SIGNATURE       * pSignature,  
                           CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

13.3.4.41 CRYPTO_ECDSA_SHA384_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-384.

Prototype

```
int CRYPTO_ECDSA_SHA384_Verify(const CRYPTO_EC_CURVE * pCurve,
                                const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                const U8 aMessage[], unsigned aMessageLen,
                                const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.42 CRYPTO_ECDSA_SHA512_224_Sign()

Description

Sign a message using the given ECDSA-SHA-512/224.

Prototype

```
int CRYPTO_ECDSA_SHA512_224_Sign(const CRYPTO_EC_CURVE * pParams,  
                                   const CRYPTO_ECDSA_PRIVATE_KEY * pKey,  
                                   const U8 aMessage[],  
                                   unsigned NumBytesMessage,  
                                   CRYPTO_ECDSA_SIGNATURE * pSignature,  
                                   CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

13.3.4.43 CRYPTO_ECDSA_SHA512_224_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512/224.

Prototype

```
int CRYPTO_ECDSA_SHA512_224_Verify(const CRYPTO_EC_CURVE * pCurve,
                                      const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                      const U8               aMessage[],
                                      unsigned              MessageLen,
                                      const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                      CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.44 CRYPTO_ECDSA_SHA512_256_Sign()

Description

Sign a message using the given ECDSA-SHA-512/256.

Prototype

```
int CRYPTO_ECDSA_SHA512_256_Sign(const CRYPTO_EC_CURVE * pParams,
                                    const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                    const U8 aMessage[],
                                    unsigned NumBytesMessage,
                                    CRYPTO_ECDSA_SIGNATURE * pSignature,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.45 CRYPTO_ECDSA_SHA512_256_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512/256.

Prototype

```
int CRYPTO_ECDSA_SHA512_256_Verify(const CRYPTO_EC_CURVE          * pCurve,
                                      const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                      const U8                      aMessage[],           MessageLen,
                                      unsigned                     * pSignature,
                                      const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                      CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.46 CRYPTO_ECDSA_SHA512_Sign()

Description

Sign a message using the given ECDSA-SHA-512.

Prototype

```
int CRYPTO_ECDSA_SHA512_Sign(const CRYPTO_EC_CURVE          * pParams,
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                           const U8                      aMessage[],
                           unsigned                     NumBytesMessage,
                           CRYPTO_ECDSA_SIGNATURE        * pSignature,
                           CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.47 CRYPTO_ECDSA_SHA512_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA-512.

Prototype

```
int CRYPTO_ECDSA_SHA512_Verify(const CRYPTO_EC_CURVE * pCurve,
                                const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                const U8 aMessage[], unsigned aMessageLen,
                                const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.48 CRYPTO_ECDSA_SHA3_224_Sign()

Description

Sign a message using the given ECDSA-SHA3-224.

Prototype

```
int CRYPTO_ECDSA_SHA3_224_Sign(const CRYPTO_EC_CURVE          * pParams,
                                const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                const U8                      aMessage[ ],
                                unsigned                     NumBytesMessage,
                                CRYPTO_ECDSA_SIGNATURE       * pSignature,
                                CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.49 CRYPTO_ECDSA_SHA3_224_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-224.

Prototype

```
int CRYPTO_ECDSA_SHA3_224_Verify(const CRYPTO_EC_CURVE          * pCurve,
                                    const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                    const U8                      aMessage[ ],
                                    unsigned                     MessageLen,
                                    const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                    CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.50 CRYPTO_ECDSA_SHA3_256_Sign()

Description

Sign a message using the given ECDSA-SHA3-256.

Prototype

```
int CRYPTO_ECDSA_SHA3_256_Sign(const CRYPTO_EC_CURVE * pParams,  
                                const CRYPTO_ECDSA_PRIVATE_KEY * pKey,  
                                const U8 aMessage[ ],  
                                unsigned NumBytesMessage,  
                                CRYPTO_ECDSA_SIGNATURE * pSignature,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.51 CRYPTO_ECDSA_SHA3_256_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-256.

Prototype

```
int CRYPTO_ECDSA_SHA3_256_Verify(const CRYPTO_EC_CURVE          * pCurve,
                                    const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                    const U8                      aMessage[ ],
                                    unsigned                     MessageLen,
                                    const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                    CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.52 CRYPTO_ECDSA_SHA3_384_Sign()

Description

Sign a message using the given ECDSA-SHA3-384.

Prototype

```
int CRYPTO_ECDSA_SHA3_384_Sign(const CRYPTO_EC_CURVE * pParams,
                                const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                const U8 aMessage[ ],
                                unsigned NumBytesMessage,
                                CRYPTO_ECDSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.53 CRYPTO_ECDSA_SHA3_384_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-384.

Prototype

```
int CRYPTO_ECDSA_SHA3_384_Verify(const CRYPTO_EC_CURVE          * pCurve,
                                    const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                    const U8                      aMessage[ ],
                                    unsigned                     MessageLen,
                                    const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                    CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.54 CRYPTO_ECDSA_SHA3_512_Sign()

Description

Sign a message using the given ECDSA-SHA3-512.

Prototype

```
int CRYPTO_ECDSA_SHA3_512_Sign(const CRYPTO_EC_CURVE * pParams,
                                 const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                 const U8 aMessage[ ],
                                 unsigned NumBytesMessage,
                                 CRYPTO_ECDSA_SIGNATURE * pSignature,
                                 CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pParams	ECDSA domain parameters.
pKey	User's private key for signing.
aMessage	Message to sign.
NumBytesMessage	Number of bytes in message.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code.
- ≥ 0 Success.

13.3.4.55 CRYPTO_ECDSA_SHA3_512_Verify()

Description

Verify the signature of a message message using the given ECDSA domain parameters and public key using ECDSA-SHA3-512.

Prototype

```
int CRYPTO_ECDSA_SHA3_512_Verify(const CRYPTO_EC_CURVE          * pCurve,
                                    const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                                    const U8                      aMessage[ ],
                                    unsigned                     MessageLen,
                                    const CRYPTO_ECDSA_SIGNATURE * pSignature,
                                    CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's ECDSA public key.
aMessage	Message to verify.
MessageLen	Number of bytes in message.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.4.56 CRYPTO_ECDSA_SignDigest()

Description

Sign a message using the given ECDSA domain parameters and private key, generating a signature.

Prototype

```
int CRYPTO_ECDSA_SignDigest(const CRYPTO_EC_CURVE * pCurve,
                           const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                           const U8 * pDigest,
                           unsigned DigestLen,
                           CRYPTO_ECDSA_SIGNATURE * pSignature,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	Pointer to elliptic curve used for signing.
pKey	Pointer to user's ECDSA private key for signing.
pDigest	Pointer to digest to sign.
DigestLen	Octet length of the digest.
pSignature	Generated signature of digest.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

13.3.4.57 CRYPTO_ECDSA_SignDigestWithK()

Description

Sign a message using the given ECDSA domain parameters and private key, and a predetermined value of K, generating a signature. This is primarily of use when running the ECDSA test vectors or when you compute K deterministically from the message, such as in [RFC6979].

Prototype

```
int CRYPTO_ECDSA_SignDigestWithK(const CRYPTO_EC_CURVE           * pCurve,
                                  const CRYPTO_ECDSA_PRIVATE_KEY * pKey,
                                  const U8                      * pDigest,
                                  unsigned                     DigestLen,
                                  const CRYPTO_MPI              * pK,
                                  CRYPTO_ECDSA_SIGNATURE        * pSignature,
                                  CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pCurve	ECDH group parameters.
pKey	User's private key for signing.
pDigest	Pointer to digest to sign.
DigestLen	Octet length of the digest.
pK	Pointer to MPI containing per-message secret value.
pSignature	Pointer to object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Error status code — signing did not complete as S, R, or both are zero and a the caller should generate a new K.
- ≥ 0 Success — the computed signature is valid for the given K with both S and R nonzero.

13.3.4.58 CRYPTO_ECDSA_VerifyDigest()

Description

Verify the signature of a message using the given ECDSA domain parameters and public key. If the signature is correct and verified, CRYPTO_ECDSA_Verify() returns nonzero. If the signature is not verified, CRYPTO_ECDSA_Verify() returns zero.

Prototype

```
int CRYPTO_ECDSA_VerifyDigest(const CRYPTO_EC_CURVE * pCurve,
                               const CRYPTO_ECDSA_PUBLIC_KEY * pKey,
                               const U8 * pDigest,
                               unsigned DigestLen,
                               const CRYPTO_ECDSA_SIGNATURE * pSignature,
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pCurve	ECDSA domain parameters.
pKey	User's public key.
pDigest	Pointer to message digest octet string.
DigestLen	Octet length of the digest octet string.
pSignature	Signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error verifying digest.
- = 0 Processing successful, but signature is not verified.
- > 0 Processing successful, signature is verified.

13.3.5 Self-test API

The following table lists the ECDSA self-test API functions.

Function	Description
<code>CRYPTO_ECDSA_PKV_CAVS_SelfTest()</code>	Run ECDSA public key verification KATs from CAVS.
<code>CRYPTO_ECDSA_Sign_CAVS_SelfTest()</code>	Run ECDSA signing test vectors from CAVS.
<code>CRYPTO_ECDSA_Verify_CAVS_SelfTest()</code>	Run ECDSA verification KATs CAVS.
<code>CRYPTO_ECDSA_RFC6979_SelfTest()</code>	Run ECDSA KATs from RFC6979.

13.3.5.1 CRYPTO_ECDSA_PKV_CAVS_SelfTest()

Description

Run ECDSA public key verification KATs from CAVS.

Prototype

```
void CRYPTO_ECDSA_PKV_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.3.5.2 CRYPTO_ECDSA_Sign_CAVS_SelfTest()

Description

Run ECDSA signing test vectors from CAVS.

Prototype

```
void CRYPTO_ECDSA_Sign_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.3.5.3 CRYPTO_ECDSA_Verify_CAVS_SelfTest()

Description

Run ECDSA verification KATs CAVS.

Prototype

```
void CRYPTO_ECDSA_Verify_CAVS_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.3.5.4 CRYPTO_ECDSA_RFC6979_SelfTest()

Description

Run ECDSA KATs from RFC6979.

Prototype

```
void CRYPTO_ECDSA_RFC6979_SelfTest(const CRYPTO_SELFTEST_API * pAPI,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.3.5.5 CRYPTO_Bench_ECDSA.c

This application benchmarks the configured performance of ECDSA sign and verify for various elliptic curves.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
ECDSA Sign and Verify Benchmark compiled Mar 19 2018 16:31:50
```

```
Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed                  = 200.000 MHz
Config: Static heap size                = 4440 bytes
Config: CRYPTO_VERSION                 = 22400 [2.24]
Config: CRYPTO_MPI_BITS_PER_LIMB       = 32
Config: CRYPTO_CONFIG_ECDSA_TWIN_MULTIPLY = 1
```

Curve	Sign ms	Sign bytes	Verify ms	Verify bytes
secp192r1	23.18	1152	21.77	1920
secp192k1	32.31	1152	30.65	1920
secp224r1	26.14	1296	25.39	2160
secp224k1	42.29	1296	39.96	2160
secp256r1	38.98	1440	36.54	2400
secp256k1	52.79	1440	51.35	2400
secp384r1	67.65	2016	63.50	3360
secp511r1	119.44	2664	110.38	4440
brainpoolP160r1	25.18	1008	23.50	1680
brainpoolP160t1	23.28	1008	22.32	1680
brainpoolP192r1	33.94	1152	32.12	1920
brainpoolP192t1	31.32	1152	28.92	1920
brainpoolP224r1	44.54	1296	40.93	2160
brainpoolP224t1	41.26	1296	39.84	2160
brainpoolP256r1	57.62	1440	54.50	2400
brainpoolP256t1	52.49	1440	49.49	2400
brainpoolP320r1	87.41	1728	82.31	2880
brainpoolP320t1	80.14	1728	74.44	2880
brainpoolP384r1	134.37	2016	124.92	3360
brainpoolP384t1	123.46	2016	116.16	3360
brainpoolP512r1	247.18	2592	229.29	4320
brainpoolP512t1	224.18	2592	208.12	4320

```
Benchmark complete
```

Complete listing

```
*****
*          (c) SEGGER Microcontroller GmbH           *
*          The Embedded Experts                  *
*          www.segger.com                      *
*****  

----- END-OF-HEADER -----  

File      : CRYPTO_Bench_ECDSA.c
Purpose   : Benchmark ECDSA sign and verify.  

*/  

*****  

*  

*      #include section  

*  

*****  

*/  

#include "CRYPTO.h"
#include "SEGGER_MEM.h"
#include "SEGGER_SYS.h"  

*****  

*  

*      Defines, configurable  

*  

*****  

*/  

#define MAX_CHUNKS          30 // For twin multiplication  

*****  

*  

*      Defines, fixed  

*  

*****  

*/  

#define CRYPTO_ASSERT(X)      { if (!(X)) { CRYPTO_PANIC(); } } // I know this is low-rent
#define CRYPTO_CHECK(X)        /*lint -e{717,801,9036}
/* do { if ((Status = (X)) < 0) goto Finally; } while (0)  

*****  

*  

*      Local types  

*  

*****  

*/  

// Maximum prime size is 521 bits, but require additional 63 bits
// for underlying fast prime field reduction.
typedef CRYPTO_MPI_LIMB MPI_UNIT[CRYPTO_MPI_LIMBS_REQUIRED(2*521+63)+2];  

*****  

*  

*      Static const data  

*  

*****  

*/  

static const U8 _aDigest[32] = { ' ', 'S', 'E', 'G', 'G', 'E', 'R', ' ',  

                                ' ', 'S', 'E', 'G', 'G', 'E', 'R', ' ',  

                                ' ', 'S', 'E', 'G', 'G', 'E', 'R', ' ',  

                                ' ', 'S', 'E', 'G', 'G', 'E', 'R', ' ' };  

*****  

*  

*      Static data  

*  

*****  

*/  

static MPI_UNIT          _aUnits[MAX_CHUNKS];
static SEGGER_MEM_CONTEXT _MemContext;
static SEGGER_MEM_SELFTEST_HEAP _Heap;  

*****  

*  

*      Static code  

*  

*****  

*/
```

```

/*
 *     _ConvertTicksToSeconds()
 *
 * Function description
 *     Convert ticks to seconds.
 *
 * Parameters
 *     Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
 *
 * Return value
 *     Number of seconds corresponding to tick.
 */
static float _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0f;
}

/*
 *     _BenchmarkECDSASign()
 *
 * Function description
 *     Benchmark ECDSA sign.
 *
 * Parameters
 *     pCurve - Pointer to elliptic curve.
 */
static void _BenchmarkECDSASign(const CRYPTO_EC_CURVE *pCurve) {
    CRYPTO_ECDSA_PRIVATE_KEY Private;
    CRYPTO_ECDSA_PUBLIC_KEY Public;
    CRYPTO_ECDSA_SIGNATURE Signature;
    U64 OneSecond;
    U64 T0;
    U64 Elapsed;
    int Loops;
    int Status;
    unsigned UnitSize;
    unsigned PeakBytes;
    float Time;
    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops = 0;
    Elapsed = 0;
    PeakBytes = 0;
    UnitSize = CRYPTO_MPI_BYTES_REQUIRED(2*CRYPTO_MPI_BitCount(&pCurve-
>P)+63) + 2*CRYPTO_MPI_BYTES_PER_LIMB;
    //
    CRYPTO_ECDSA_InitPrivateKey(&Private, &_MemContext);
    CRYPTO_ECDSA_InitPublicKey (&Public, &_MemContext);
    CRYPTO_ECDSA_InitSignature (&Signature, &_MemContext);
    //
    CRYPTO_ECDSA_GenKeys (pCurve, &Private, &Public, &_MemContext);
    //
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        //
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        //
        CRYPTO_CHECK(CRYPTO_ECDSA_SignDigest(pCurve, &Private, &_aDigest[0], sizeof(_aDigest), &Signature, &_MemContext));
        if (Status == 0) {
            SEGGER_SYS_IO_Printf("ERROR - Did not sign digest\n");
            SEGGER_SYS_OS_Halt(100);
        }
        //
        PeakBytes = SEGGER_MAX(PeakBytes, _Heap.Stats.NumInUseMax * UnitSize);
        //
        CRYPTO_ECDSA_KillSignature(&Signature);
        //
        Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
        ++Loops;
    } while (Status >= 0 && Elapsed < OneSecond);
    //
Finally:
    CRYPTO_ECDSA_KillPrivateKey(&Private);
    CRYPTO_ECDSA_KillPublicKey (&Public);
    CRYPTO_ECDSA_KillSignature (&Signature);
    //
    if (Status < 0 || Loops == 0) {
        SEGGER_SYS_IO_Printf("%10s |", "-Fail-");
    } else {
        Loops *= 2; // Two agreements per loop
        Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
        SEGGER_SYS_IO_Printf("%10.2f |", Time);
    }
}

```

```

        SEGGER_SYS_IO_Printf("%10d | ", PeakBytes);
    }

} ****
*
*     _BenchmarkECDSAVerify()
*
* Function description
*     Benchmark ECDSA verify.
*
* Parameters
*     pCurve - Pointer to elliptic curve.
*/
static void _BenchmarkECDSAVerify(const CRYPTO_EC_CURVE *pCurve) {
    CRYPTO_ECDSA_PRIVATE_KEY Private;
    CRYPTO_ECDSA_PUBLIC_KEY Public;
    CRYPTO_ECDSA_SIGNATURE Signature;
    U64 OneSecond;
    U64 T0;
    U64 Elapsed;
    int Loops;
    int Status;
    unsigned PeakBytes;
    unsigned UnitSize;
    float Time;

    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops = 0;
    Elapsed = 0;
    PeakBytes = 0;
    UnitSize = CRYPTO_MPI_BYTES_REQUIRED(2*CRYPTO_MPI_BitCount(&pCurve-
>P)+63) + 2*CRYPTO_MPI_BYTES_PER_LIMB;
    //
    CRYPTO_ECDSA_InitPrivateKey(&Private, &_MemContext);
    CRYPTO_ECDSA_InitPublicKey (&Public, &_MemContext);
    CRYPTO_ECDSA_InitSignature (&Signature, &_MemContext);
    //
    CRYPTO_ECDSA_GenKeys (pCurve, &Private, &Public, &_MemContext);
    //
    CRYPTO_CHECK(CRYPTO_ECDSA_SignDigest(pCurve, &Private, &_aDigest[0], sizeof(_aDigest), &Signature, &_MemContext));
    if (Status == 0) {
        SEGGER_SYS_IO_Printf("ERROR - Did not sign digest\n");
        SEGGER_SYS_OS_Halt(100);
    }
    //
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        //
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        //

        CRYPTO_CHECK(CRYPTO_ECDSA_VerifyDigest(pCurve, &Public, &_aDigest[0], sizeof(_aDigest), &Signature, &_MemContext));
        if (Status == 0) {
            SEGGER_SYS_IO_Printf("ERROR - Did not verify digest\n");
            SEGGER_SYS_OS_Halt(100);
        }
        //
        PeakBytes = SEGGER_MAX(PeakBytes, _Heap.Stats.NumInUseMax * UnitSize);
        //
        Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
        ++Loops;
    } while (Status >= 0 && Elapsed < OneSecond);
    //
Finally:
    CRYPTO_ECDSA_KillPrivateKey(&Private);
    CRYPTO_ECDSA_KillPublicKey (&Public);
    CRYPTO_ECDSA_KillSignature (&Signature);
    //
    if (Status < 0 || Loops == 0) {
        SEGGER_SYS_IO_Printf("%10s | ", "-Fail-");
    } else {
        Loops *= 2; // Two agreements per loop
        Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
        SEGGER_SYS_IO_Printf("%10.2f | ", Time);
        SEGGER_SYS_IO_Printf("%10d | ", PeakBytes);
    }
}

} ****
*
*     _BenchmarkECDSA()
*
* Function description
*     Benchmark ECDSA sign and verify.

```

```

/*
 * Parameters
 *   pCurve - Pointer to elliptic curve.
 */
static void _BenchmarkECDSA(const CRYPTO_EC_CURVE *pCurve) {
    SEGGER_SYS_IO_Printf(" | %-16s | ", pCurve->aCurveName);
    _BenchmarkECDSASign (pCurve);
    _BenchmarkECDSAVerify(pCurve);
    SEGGER_SYS_IO_Printf("\n");
}

***** Public code *****
*/
***** MainTask() *****
* Function description
* Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    SEGGER_MEM_SELFTEST_HEAP_Init(&_MemContext, &_Heap, _aUnits, MAX_CHUNKS, sizeof(MPI_UNIT));
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("ECDSA Sign and Verify Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config: Static heap size = %u bytes\n", sizeof(_aUnits));
    SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u\n%s\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config: CRYPTO_MPI_BITS_PER_LIMB = %u\n", CRYPTO_MPI_BITS_PER_LIMB);
    SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_ECDSA_TWIN_MULTIPLY = %u\n", CRYPTO_CONFIG_ECDSA_TWIN_MULTIPLY);
    SEGGER_SYS_IO_Printf("\n");
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+-----+-----\n");
    SEGGER_SYS_IO_Printf(" |           |     Sign |     Sign |   Verify |   Verify | \n");
    SEGGER_SYS_IO_Printf(" | Curve   |       ms |   bytes |       ms |   bytes | \n");
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+-----+-----\n");
    //
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp192r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp192k1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp224r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp224k1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp256r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp256k1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp384r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_secp511r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP160r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP160t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP192r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP192t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP224r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP224t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP256r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP256t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP320r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP320t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP384r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP384t1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP512r1);
    _BenchmarkECDSA(&CRYPTO_EC_CURVE_briapoolP512t1);
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+-----+-----\n");
    SEGGER_SYS_IO_Printf("\n");
    //
    SEGGER_SYS_IO_Printf("Benchmark complete\n");
    SEGGER_SYS_OS_PauseBeforeHalt();
    SEGGER_SYS_OS_Halt(0);
}

***** End of file *****

```

13.4 EdDSA

13.4.1 Management

The following table lists the EdDSA key management API.

Function	Description
<code>CRYPTO_EdDSA_InitPublicKey()</code>	Initialize EdDSA public key.
<code>CRYPTO_EdDSA_InitPrivateKey()</code>	Initialize EdDSA private key.
<code>CRYPTO_EdDSA_InitSignature()</code>	Initialize EdDSA signature.
<code>CRYPTO_EdDSA_KillPublicKey()</code>	Destroy EdDSA public key.
<code>CRYPTO_EdDSA_KillPrivateKey()</code>	Destroy EdDSA private key.
<code>CRYPTO_EdDSA_KillSignature()</code>	Destroy EdDSA signature.

13.4.1.1 CRYPTO_EdDSA_InitPrivateKey()

Description

Initialize EdDSA private key.

Prototype

```
void CRYPTO_EdDSA_InitPrivateKey(CRYPTO_EdDSA_PRIVATE_KEY * pSelf,  
                                 CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to EdDSA private key.
pMem	Pointer to memory allocator to use for key storage.

13.4.1.2 CRYPTO_EdDSA_InitPublicKey()

Description

Initialize EdDSA public key.

Prototype

```
void CRYPTO_EdDSA_InitPublicKey(CRYPTO_EdDSA_PUBLIC_KEY * pSelf,  
                                CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to EdDSA public key.
pMem	Pointer to memory allocator to use for key storage.

13.4.1.3 CRYPTO_EdDSA_InitSignature()

Description

Initialize EdDSA signature.

Prototype

```
void CRYPTO_EdDSA_InitSignature(CRYPTO_EdDSA_SIGNATURE * pSelf,  
                                CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to EdDSA signature.
pMem	Pointer to memory allocator to use for key storage.

13.4.1.4 CRYPTO_EdDSA_KillPrivateKey()

Description

Destroy EdDSA private key.

Prototype

```
void CRYPTO_EdDSA_KillPrivateKey(CRYPTO_EdDSA_PRIVATE_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to EdDSA private key.

13.4.1.5 CRYPTO_EdDSA_KillPublicKey()

Description

Destroy EdDSA public key.

Prototype

```
void CRYPTO_EdDSA_KillPublicKey(CRYPTO_EdDSA_PUBLIC_KEY * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to EdDSA public key.

13.4.1.6 CRYPTO_EdDSA_KillSignature()

Description

Destroy EdDSA signature.

Prototype

```
void CRYPTO_EdDSA_KillSignature(CRYPTO_EdDSA_SIGNATURE * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to EdDSA signature.

13.4.2 Ed25519 Type-safe API

The following table lists the Ed25519 API.

Function	Description
Sign	
<code>CRYPTO_EdDSA_Ed25519_Sign()</code>	Sign message.
<code>CRYPTO_EdDSA_Ed25519_SignEx()</code>	Sign message, with context.
<code>CRYPTO_EdDSA_Ed25519_SignDigest()</code>	Sign message digest.
Verify	
<code>CRYPTO_EdDSA_Ed25519_Verify()</code>	Verify message.
<code>CRYPTO_EdDSA_Ed25519_VerifyEx()</code>	Verify message, with context.
<code>CRYPTO_EdDSA_Ed25519_VerifyDigest()</code>	Verify message digest.
Keys	
<code>CRYPTO_EdDSA_Ed25519_GenKeys()</code>	Generate key pair, random seed.
<code>CRYPTO_EdDSA_Ed25519_GenKeysEx()</code>	Generate key pair, explicit seed.
<code>CRYPTO_EdDSA_Ed25519_CalcPublicKey()</code>	Compute public key from private key.
I/O	
<code>CRYPTO_EdDSA_Ed25519_RdPublicKey()</code>	Read binary form of public key.
<code>CRYPTO_EdDSA_Ed25519_WrPublicKey()</code>	Write binary form of public key.
<code>CRYPTO_EdDSA_Ed25519_RdPrivateKey()</code>	Read binary form of private key.
<code>CRYPTO_EdDSA_Ed25519_WrPrivateKey()</code>	Write binary form of private key.
<code>CRYPTO_EdDSA_Ed25519_RdSignature()</code>	Read binary form of signature.
<code>CRYPTO_EdDSA_Ed25519_WrSignature()</code>	Write binary form of signature.

13.4.2.1 CRYPTO_EdDSA_Ed25519_Sign()

Description

Sign message.

Prototype

```
int CRYPTO_EdDSA_Ed25519_Sign(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                const U8 * pMessage,  
                                unsigned MessageLen,  
                                CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.2.2 CRYPTO_EdDSA_Ed25519_SignEx()

Description

Sign message, with context.

Prototype

```
int CRYPTO_EdDSA_Ed25519_SignEx(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
                                  const U8                           * pMessage,
                                  unsigned                         MessageLen,
                                  const U8                           * pContext,
                                  unsigned                         ContextLen,
                                  CRYPTO_EdDSA_SIGNATURE        * pSignature,
                                  CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.2.3 CRYPTO_EdDSA_Ed25519_SignDigest()

Description

Sign message digest.

Prototype

```
int CRYPTO_EdDSA_Ed25519_SignDigest(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                     const U8 * pDigest,  
                                     unsigned DigestLen,  
                                     CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pDigest	Pointer to message digest.
DigestLen	Octet length of the message digest.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.2.4 CRYPTO_EdDSA_Ed25519_Verify()

Description

Verify message.

Prototype

```
int CRYPTO_EdDSA_Ed25519_Verify(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                  const U8 * pMessage,  
                                  unsigned MessageLen,  
                                  const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pMessage	Pointer to message to verify.
MessageLen	Octet length of the message to verify.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.2.5 CRYPTO_EdDSA_Ed25519_VerifyEx()

Description

Verify message, with context.

Prototype

```
int CRYPTO_EdDSA_Ed25519_VerifyEx(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,
                                     const U8                           * pMessage,
                                     unsigned                         MessageLen,
                                     const U8                           * pContext,
                                     unsigned                         ContextLen,
                                     const CRYPTO_EdDSA_SIGNATURE   * pSignature,
                                     CRYPTO_MEM_CONTEXT            * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pMessage	Pointer to message to verify.
MessageLen	Octet length of the message to verify.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.2.6 CRYPTO_EdDSA_Ed25519_VerifyDigest()

Description

Verify message digest.

Prototype

```
int CRYPTO_EdDSA_Ed25519_VerifyDigest(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                       const U8 * pDigest,  
                                       unsigned DigestLen,  
                                       const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pDigest	Pointer to message to verify.
DigestLen	Octet length of the message to verify.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.2.7 CRYPTO_EdDSA_Ed25519_CalcPublicKey()

Description

Compute public key from private key.

Prototype

```
int CRYPTO_EdDSA_Ed25519_CalcPublicKey
    (const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
     CRYPTO_EdDSA_PUBLIC_KEY    * pPublicKey,
     CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to private key.
pPublicKey	Pointer to MPI that will receives the public key.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error indication.
≥ 0 Success.

13.4.2.8 CRYPTO_EdDSA_Ed25519_GenKeys()

Description

Generate key pair, random seed.

Prototype

```
int CRYPTO_EdDSA_Ed25519_GenKeys(CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                  CRYPTO_EdDSA_PUBLIC_KEY  * pPublicKey,  
                                  CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to object that receives the private key.
pPublicKey	Pointer to object that receives the public key.
pMem	Pointer to memory allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

The seed is initialized from the installed random number generator.

13.4.2.9 CRYPTO_EdDSA_Ed25519_GenKeysEx()

Description

Generate key pair, explicit seed.

Prototype

```
int CRYPTO_EdDSA_Ed25519_GenKeysEx(          CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
                                             CRYPTO_EdDSA_PUBLIC_KEY  * pPublicKey,
                                             const U8                  * pSeed,
                                             CRYPTO_MEM_CONTEXT       * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to object that receives the private key.
pPublicKey	Pointer to object that receives the public key.
pSeed	Pointer to 32-octet seed.
pMem	Pointer to memory allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.2.10 CRYPTO_EdDSA_Ed25519_RdSignature()

Description

Read binary form of signature.

Prototype

```
int CRYPTO_EdDSA_Ed25519_RdSignature( CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                      const U8           * pInput);
```

Parameters

Parameter	Description
pSignature	Pointer to EdDSA signature.
pInput	Pointer to octet string that contains the binary form of the Ed25519 signature; 64 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.2.11 CRYPTO_EdDSA_Ed25519_WrSignature()

Description

Write binary form of signature.

Prototype

```
void CRYPTO_EdDSA_Ed25519_WrSignature(const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                         U8                         * pOutput);
```

Parameters

Parameter	Description
pSignature	Pointer to EdDSA signature.
pOutput	Pointer to octet string that receives the binary form of the Ed25519 signature; 64 octets.

13.4.2.12 CRYPTO_EdDSA_Ed25519_RdPublicKey()

Description

Read binary form of public key.

Prototype

```
int CRYPTO_EdDSA_Ed25519_RdPublicKey(          CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                         const U8             * pInput);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key.
pInput	Pointer to octet string that contains the binary form of the public key; 32 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.2.13 CRYPTO_EdDSA_Ed25519_WrPublicKey()

Description

Write binary form of public key.

Prototype

```
void CRYPTO_EdDSA_Ed25519_WrPublicKey(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                         U8 * pOutput);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key.
pOutput	Pointer to octet string that receives public key in external form; 32 octets.

13.4.2.14 CRYPTO_EdDSA_Ed25519_RdPrivateKey()

Description

Read binary form of private key.

Prototype

```
int CRYPTO_EdDSA_Ed25519_RdPrivateKey( CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                         const U8 * pInput);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key.
pInput	Pointer to octet string that contains the binary form of the private key; 32 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.2.15 CRYPTO_EdDSA_Ed25519_WrPrivateKey()

Description

Write binary form of private key.

Prototype

```
void CRYPTO_EdDSA_Ed25519_WrPrivateKey
    (const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
     U8                                * pOutput);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key.
pOutput	Pointer to octet string that receives private key in external form; 32 octets.

13.4.3 Ed448 Type-safe API

The following table lists the Ed448 API.

Function	Description
Sign	
<code>CRYPTO_EdDSA_Ed448_Sign()</code>	Sign message.
<code>CRYPTO_EdDSA_Ed448_SignEx()</code>	Sign message, with context.
<code>CRYPTO_EdDSA_Ed448_SignDigest()</code>	Sign message digest.
<code>CRYPTO_EdDSA_Ed448_SignDigestEx()</code>	Sign message digest, with context.
Verify	
<code>CRYPTO_EdDSA_Ed448_Verify()</code>	Verify message.
<code>CRYPTO_EdDSA_Ed448_VerifyEx()</code>	Verify message, with context.
<code>CRYPTO_EdDSA_Ed448_VerifyDigest()</code>	Verify message digest.
<code>CRYPTO_EdDSA_Ed448_VerifyDigestEx()</code>	Verify message digest, with context.
Keys	
<code>CRYPTO_EdDSA_Ed448_GenKeys()</code>	Generate key pair, random seed.
<code>CRYPTO_EdDSA_Ed448_GenKeysEx()</code>	Generate key pair, explicit seed.
<code>CRYPTO_EdDSA_Ed448_CalcPublicKey()</code>	Compute public key from private key.
I/O	
<code>CRYPTO_EdDSA_Ed448_RdPublicKey()</code>	Read binary form of public key.
<code>CRYPTO_EdDSA_Ed448_WrPublicKey()</code>	Write binary form of public key.
<code>CRYPTO_EdDSA_Ed448_RdPrivateKey()</code>	Read binary form of private key.
<code>CRYPTO_EdDSA_Ed448_WrPrivateKey()</code>	Write binary form of private key.
<code>CRYPTO_EdDSA_Ed448_RdSignature()</code>	Read binary form of signature.
<code>CRYPTO_EdDSA_Ed448_WrSignature()</code>	Write binary form of signature.

13.4.3.1 CRYPTO_EdDSA_Ed448_Sign()

Description

Sign message.

Prototype

```
int CRYPTO_EdDSA_Ed448_Sign(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                           const U8 * pMessage,  
                           unsigned MessageLen,  
                           CRYPTO_EdDSA_SIGNATURE * pSignature,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.3.2 CRYPTO_EdDSA_Ed448_SignEx()

Description

Sign message, with context.

Prototype

```
int CRYPTO_EdDSA_Ed448_SignEx(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
                                const U8               * pMessage,
                                unsigned             MessageLen,
                                const U8               * pContext,
                                unsigned             ContextLen,
                                CRYPTO_EdDSA_SIGNATURE * pSignature,
                                CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pMessage	Pointer to message to sign.
MessageLen	Octet length of the message to sign.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.3.3 CRYPTO_EdDSA_Ed448_SignDigest()

Description

Sign message digest.

Prototype

```
int CRYPTO_EdDSA_Ed448_SignDigest(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                    const U8  
                                    unsigned  
                                    CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pDigest	Pointer to message digest.
DigestLen	Octet length of the message digest.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.3.4 CRYPTO_EdDSA_Ed448_SignDigestEx()

Description

Sign message digest, with context.

Prototype

```
int CRYPTO_EdDSA_Ed448_SignDigestEx(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,
                                      const U8                           * pDigest,
                                      unsigned                         DigestLen,
                                      const U8                           * pContext,
                                      unsigned                         ContextLen,
                                      CRYPTO_EdDSA_SIGNATURE        * pSignature,
                                      CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key of the key pair.
pDigest	Pointer to message digest.
DigestLen	Octet length of the message digest.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to initialized object that receives the signature.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Signature generated.

13.4.3.5 CRYPTO_EdDSA_Ed448_Verify()

Description

Verify message.

Prototype

```
int CRYPTO_EdDSA_Ed448_Verify(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                const U8 * pMessage,  
                                unsigned MessageLen,  
                                const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pMessage	Pointer to message to verify.
MessageLen	Octet length of the message to verify.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.3.6 CRYPTO_EdDSA_Ed448_VerifyEx()

Description

Verify message, with context.

Prototype

```
int CRYPTO_EdDSA_Ed448_VerifyEx(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,
                                  const U8                         * pMessage,
                                  unsigned                        MessageLen,
                                  const U8                         * pContext,
                                  unsigned                        ContextLen,
                                  const CRYPTO_EdDSA_SIGNATURE * pSignature,
                                  CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pMessage	Pointer to message to verify.
MessageLen	Octet length of the message to verify.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.3.7 CRYPTO_EdDSA_Ed448_VerifyDigest()

Description

Verify message digest.

Prototype

```
int CRYPTO_EdDSA_Ed448_VerifyDigest(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                     const U8 * pDigest,  
                                     unsigned DigestLen,  
                                     const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pDigest	Pointer to message to verify.
DigestLen	Octet length of the message to verify.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.3.8 CRYPTO_EdDSA_Ed448_VerifyDigestEx()

Description

Verify message digest, with context.

Prototype

```
int CRYPTO_EdDSA_Ed448_VerifyDigestEx(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,
                                         const U8                           * pDigest,
                                         unsigned                         DigestLen,
                                         const U8                           * pContext,
                                         unsigned                         ContextLen,
                                         const CRYPTO_EdDSA_SIGNATURE * pSignature,
                                         CRYPTO_MEM_CONTEXT    * pMem);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key of the key pair.
pDigest	Pointer to message to verify.
DigestLen	Octet length of the message to verify.
pContext	Pointer to context octet string.
ContextLen	Octet length of the context octet string.
pSignature	Pointer to signature to verify.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Signature is not valid.
- > 0 Signature is valid.

13.4.3.9 CRYPTO_EdDSA_Ed448_GenKeys()

Description

Generate key pair, random seed.

Prototype

```
int CRYPTO_EdDSA_Ed448_GenKeys(CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                CRYPTO_EdDSA_PUBLIC_KEY  * pPublicKey,  
                                CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to object that receives the private key.
pPublicKey	Pointer to object that receives the public key.
pMem	Pointer to memory allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

The seed is initialized from the installed random number generator.

13.4.3.10 CRYPTO_EdDSA_Ed448_GenKeysEx()

Description

Generate key pair, explicit seed.

Prototype

```
int CRYPTO_EdDSA_Ed448_GenKeysEx( CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                    CRYPTO_EdDSA_PUBLIC_KEY  * pPublicKey,  
                                    const U8                * pSeed,  
                                    CRYPTO_MEM_CONTEXT      * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to object that receives the private key.
pPublicKey	Pointer to object that receives the public key.
pSeed	Pointer to 57-octet seed.
pMem	Pointer to memory allocator to use for temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.3.11 CRYPTO_EdDSA_Ed448_CalcPublicKey()

Description

Compute public key from private key.

Prototype

```
int CRYPTO_EdDSA_Ed448_CalcPublicKey(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                      CRYPTO_EdDSA_PUBLIC_KEY   * pPublicKey,  
                                      CRYPTO_MEM_CONTEXT       * pMem);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key.
pPublicKey	Pointer to initialized object that will receive the EdDSA public key.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error indication.
≥ 0 Success.

13.4.3.12 CRYPTO_EdDSA_Ed448_RdPublicKey()

Description

Read binary form of public key.

Prototype

```
int CRYPTO_EdDSA_Ed448_RdPublicKey( CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                     const U8 * pInput);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key.
pInput	Pointer to octet string that contains the binary form of the public key; 57 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.3.13 CRYPTO_EdDSA_Ed448_WrPublicKey()

Description

Write binary form of public key.

Prototype

```
void CRYPTO_EdDSA_Ed448_WrPublicKey(const CRYPTO_EdDSA_PUBLIC_KEY * pPublicKey,  
                                     U8 * pOutput);
```

Parameters

Parameter	Description
pPublicKey	Pointer to EdDSA public key.
pOutput	Pointer to octet string that receives public key in external form; 57 octets.

13.4.3.14 CRYPTO_EdDSA_Ed448_RdPrivateKey()

Description

Read binary form of private key.

Prototype

```
int CRYPTO_EdDSA_Ed448_RdPrivateKey( CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                     const U8 * pInput);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key.
pInput	Pointer to octet string that contains the binary form of the private key; 57 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.3.15 CRYPTO_EdDSA_Ed448_WrPrivateKey()

Description

Write binary form of private key.

Prototype

```
void CRYPTO_EdDSA_Ed448_WrPrivateKey(const CRYPTO_EdDSA_PRIVATE_KEY * pPrivateKey,  
                                      U8                      * pOutput);
```

Parameters

Parameter	Description
pPrivateKey	Pointer to EdDSA private key.
pOutput	Pointer to octet string that receives private key in external form; 57 octets.

13.4.3.16 CRYPTO_EdDSA_Ed448_RdSignature()

Description

Read binary form of signature.

Prototype

```
int CRYPTO_EdDSA_Ed448_RdSignature( CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                     const U8 * pInput);
```

Parameters

Parameter	Description
pSignature	Pointer to EdDSA signature.
pInput	Pointer to octet string that contains the binary form of the Ed448 signature; 114 octets.

Return value

< 0 Processing error.
≥ 0 Success.

13.4.3.17 CRYPTO_EdDSA_Ed448_WrSignature()

Description

Write binary form of signature.

Prototype

```
void CRYPTO_EdDSA_Ed448_WrSignature(const CRYPTO_EdDSA_SIGNATURE * pSignature,  
                                     U8 * pOutput);
```

Parameters

Parameter	Description
pSignature	Pointer to EdDSA signature.
pOutput	Pointer to octet string that receives the binary form of the Ed448 signature; 114 octets.

13.4.4 Self-test API

The following table lists the EdDSA self-test API functions.

Function	Description
CRYPTO_EdDSA_Ed25519_Bernstein_SelfTest()	Run Ed25519 KATs from Bernstein.
CRYPTO_EdDSA_Ed25519_RFC8032_SelfTest()	Run Ed25519 self-tests from RFC 8032.
CRYPTO_EdDSA_Ed448_RFC8032_SelfTest()	Run Ed448 self-tests from RFC 8032.

13.4.4.1 CRYPTO_EdDSA_Ed25519_Bernstein_SelfTest()

Description

Run Ed25519 KATs from Bernstein.

Prototype

```
void CRYPTO_EdDSA_Ed25519_Bernstein_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

13.4.4.2 CRYPTO_EdDSA_Ed25519_RFC8032_SelfTest()

Description

Run Ed25519 self-tests from RFC 8032.

Prototype

```
void CRYPTO_EdDSA_Ed25519_RFC8032_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

13.4.4.3 CRYPTO_EdDSA_Ed448_RFC8032_SelfTest()

Description

Run Ed448 self-tests from RFC 8032.

Prototype

```
void CRYPTO_EdDSA_Ed448_RFC8032_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

13.4.5 Example applications

13.4.5.1 CRYPTO_Bench_EdDSA.c

This application benchmarks the configured performance of EdDSA for Curve25519 and Curve448.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
EdDSA Sign and Verify Benchmark compiled Mar 19 2018 16:32:53

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: Static heap size        = 3844 bytes
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_MPI_BITS_PER_LIMB = 32

+-----+-----+
| Curve | ms/Sign |
+-----+-----+
| Ed25519 |     38.45 |
| Ed448   |     68.74 |
+-----+-----+

+-----+-----+
| Curve | ms/Verify |
+-----+-----+
| Ed25519 |     87.75 |
| Ed448   |    151.18 |
+-----+-----+

Benchmark complete
```

Complete listing

```
/****************************************************************************
 * (c) SEGGER Microcontroller GmbH
 *      The Embedded Experts
 *      www.segger.com
 ****
 ----- END-OF-HEADER -----
File      : CRYPTO_Bench_EdDSA.c
Purpose   : Benchmark EdDSA sign and verify.

*/
/*
*      #include section
*
*****
*/
#include "CRYPTO.h"
#include "SEGGER_MEM.h"
#include "SEGGER_SYS.h"

/*
*      Defines, fixed
*
*****
*/
#define CRYPTO_ASSERT(X)           { if (!(X)) { CRYPTO_PANIC(); } } // I know this is low-rent
#define CRYPTO_CHECK(X)            /*lint -e{717,801,9036} */
/* do { if ((Status = (X)) < 0) goto Finally; } while (0)

*****
```

```

*      Defines, configurable
*
***** Local types *****
*/
#define MAX_CHUNKS          31

***** Static data *****
*/
static MPI_UNIT           _aUnits[MAX_CHUNKS];
static SEGGER_MEM_CONTEXT _MemContext;
static SEGGER_MEM_SELFTEST_HEAP _Heap;
static int                 _ShowMemory = 0;

***** Static code *****
*/
*      _ConvertTicksToSeconds()
*
* Function description
*   Convert ticks to seconds.
*
* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static float _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0f;
}

***** BenchmarkEd25519Sign()
*
* Function description
*   Benchmark Ed25519 sign.
*/
static void _BenchmarkEd25519Sign(void) {
    CRYPTO_EdDSA_PRIVATE_KEY Private;
    CRYPTO_EdDSA_PUBLIC_KEY Public;
    CRYPTO_EdDSA_SIGNATURE Signature;
    U8                      aMsg[] = { "SEGGER: It simply works!" };
    U64                     Limit;
    U64                     T0;
    U64                     Elapsed;
    int                     Loops;
    int                     Status;
    unsigned                PeakBytes;
    float                   Time;
    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops      = 0;
    Elapsed   = 0;
    PeakBytes = 0;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", "Ed25519");
    //
    CRYPTO_MEMSET(aMsg, 0, sizeof(aMsg));
    CRYPTO_EdDSA_InitPrivateKey(&Private,     &_MemContext);
    CRYPTO_EdDSA_InitPublicKey (&Public,       &_MemContext);
    CRYPTO_EdDSA_InitSignature(&Signature,    &_MemContext);
    //
    CRYPTO_CHECK(CRYPTO_EdDSA_Ed25519_GenKeys(&Private, &Public, &_MemContext));
}

```

```

//  

Limit = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);  

T0 = SEGGER_SYS_OS_GetTimer();  

do {  

//  

_Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;  

//  

CRYPTO_CHECK(CRYPTO_EdDSA_Ed25519_Sign(&Private, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));  

CRYPTO_EdDSA_KillSignature(&Signature);  

//  

PeakBytes = _Heap.Stats.NumInUseMax * sizeof(MPI_UNIT);  

//  

Elapsed = SEGGER_SYS_OS_GetTimer() - T0;  

++Loops;  

} while (Status >= 0 && Elapsed < Limit);  

//  

Finally:  

CRYPTO_EdDSA_KillPrivateKey(&Private);  

CRYPTO_EdDSA_KillPublicKey (&Public);  

CRYPTO_EdDSA_KillSignature (&Signature);  

//  

if (Status < 0 || Loops == 0) {  

SEGGER_SYS_IO_Printf("%13s | ", "-Fail-");  

} else if (_ShowMemory) {  

SEGGER_SYS_IO_Printf("%13d | ", PeakBytes);  

} else {  

Loops *= 2; // Two agreements per loop  

Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;  

if (_ShowMemory) {  

SEGGER_SYS_IO_Printf("%8d | ", PeakBytes);  

} else {  

SEGGER_SYS_IO_Printf("%13.2f | ", Time);  

}  

}  

SEGGER_SYS_IO_Printf("\n");
}

*****
*  

*      _BenchmarkEd448Sign()  

*  

*  Function description  

*      Benchmark Ed448 sign.  

*/
static void _BenchmarkEd448Sign(void) {
CRYPTO_EdDSA_PRIVATE_KEY Private;
CRYPTO_EdDSA_SIGNATURE Signature;
U8 aMsg[] = { "SEGGER: It simply works!" };
U64 Limit;
U64 T0;
U64 Elapsed;
int Loops;
int Status;
unsigned PeakBytes;
float Time;
//  

// Make PC-lint quiet, it's dataflow analysis provides false positives.
//  

Loops = 0;
Elapsed = 0;
PeakBytes = 0;
//  

SEGGER_SYS_IO_Printf(" | %-12s | ", "Ed448");
//  

CRYPTO_MEMSET(aMsg, 0, sizeof(aMsg));
CRYPTO_EdDSA_InitPrivateKey(&Private, &_MemContext);
CRYPTO_EdDSA_InitSignature (&Signature, &_MemContext);
//  

Limit = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
T0 = SEGGER_SYS_OS_GetTimer();
do {
//  

_Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
//  

CRYPTO_CHECK(CRYPTO_EdDSA_Ed448_Sign(&Private, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));
CRYPTO_EdDSA_KillSignature(&Signature);
//  

PeakBytes = _Heap.Stats.NumInUseMax * sizeof(MPI_UNIT);
//  

Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
++Loops;
} while (Status >= 0 && Elapsed < Limit);
//  

Finally:  

CRYPTO_EdDSA_KillPrivateKey(&Private);
CRYPTO_EdDSA_KillSignature (&Signature);
//
```

```

if (Status < 0 || Loops == 0) {
    SEGGER_SYS_IO_Printf("%13s |", "-Fail-");
} else if (_ShowMemory) {
    SEGGER_SYS_IO_Printf("%13d |", PeakBytes);
} else {
    Loops *= 2; // Two agreements per loop
    Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
    if (_ShowMemory) {
        SEGGER_SYS_IO_Printf("%8d |", PeakBytes);
    } else {
        SEGGER_SYS_IO_Printf("%13.2f |", Time);
    }
}
SEGGER_SYS_IO_Printf("\n");
}

*****
*
*     _BenchmarkEd25519Verify()
*
* Function description
*   Benchmark Ed25519 verify.
*/
static void _BenchmarkEd25519Verify(void) {
CRYPTO_EdDSA_PRIVATE_KEY Private;
CRYPTO_EdDSA_PUBLIC_KEY Public;
CRYPTO_EdDSA_SIGNATURE Signature;
U8 aMsg[] = { "SEGGER: It simply works!" };
U64 Limit;
U64 T0;
U64 Elapsed;
int Loops;
int Status;
unsigned PeakBytes;
float Time;
//
// Make PC-lint quiet, it's dataflow analysis provides false positives.
//
Loops = 0;
Elapsed = 0;
PeakBytes = 0;
//
SEGGER_SYS_IO_Printf(" | %-12s |", "Ed25519");
//
CRYPTO_MEMSET(aMsg, 0, sizeof(aMsg));
CRYPTO_EdDSA_InitPrivateKey(&Private, &_MemContext);
CRYPTO_EdDSA_InitPublicKey (&Public, &_MemContext);
CRYPTO_EdDSA_InitSignature (&Signature, &_MemContext);
//
CRYPTO_CHECK(CRYPTO_EdDSA_Ed25519_GenKeys(&Private, &Public, &_MemContext));
CRYPTO_CHECK(CRYPTO_EdDSA_Ed25519_Sign(&Private, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));
//
Limit = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
T0 = SEGGER_SYS_OS_GetTimer();
do {
    //
    _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
    //
    CRYPTO_CHECK(CRYPTO_EdDSA_Ed25519_Verify(&Public, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));
    //
    PeakBytes = _Heap.Stats.NumInUseMax * sizeof(MPI_UNIT);
    //
    Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
    ++Loops;
} while (Status >= 0 && Elapsed < Limit);
//
Finally:
CRYPTO_EdDSA_KillPrivateKey(&Private);
CRYPTO_EdDSA_KillPublicKey (&Public);
CRYPTO_EdDSA_KillSignature (&Signature);
//
if (Status < 0 || Loops == 0) {
    SEGGER_SYS_IO_Printf("%13s |", "-Fail-");
} else if (_ShowMemory) {
    SEGGER_SYS_IO_Printf("%13d |", PeakBytes);
} else {
    Loops *= 2; // Two agreements per loop
    Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
    if (_ShowMemory) {
        SEGGER_SYS_IO_Printf("%8d |", PeakBytes);
    } else {
        SEGGER_SYS_IO_Printf("%13.2f |", Time);
    }
}
SEGGER_SYS_IO_Printf("\n");
}

```

```
*****
*
*      _BenchmarkEd448Verify()
*
*  Function description
*      Benchmark Ed448 verify.
*/
static void _BenchmarkEd448Verify(void) {
    CRYPTO_EdDSA_PRIVATE_KEY Private;
    CRYPTO_EdDSA_PUBLIC_KEY Public;
    CRYPTO_EdDSA_SIGNATURE Signature;
    U8 aMsg[] = { "SEGGER: It simply works!" };
    U64 Limit;
    U64 T0;
    Elapsed;
    Loops;
    Status;
    PeakBytes;
    Time;
    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops     = 0;
    Elapsed   = 0;
    PeakBytes = 0;
    //
    SEGGER_SYS_IO_Printf("|-12s |", "Ed448");
    //
    CRYPTO_MEMSET(aMsg, 0, sizeof(aMsg));
    CRYPTO_EdDSA_InitPrivateKey(&Private, &_MemContext);
    CRYPTO_EdDSA_InitPublicKey (&Public, &_MemContext);
    CRYPTO_EdDSA_InitSignature (&Signature, &_MemContext);
    //
    CRYPTO_CHECK(CRYPTO_EdDSA_Ed448_CalcPublicKey(&Private, &Public, &_MemContext));
    CRYPTO_CHECK(CRYPTO_EdDSA_Ed448_Sign(&Private, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));
    //
    Limit = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        //
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        //
        CRYPTO_CHECK(CRYPTO_EdDSA_Ed448_Verify(&Public, &aMsg[0], sizeof(aMsg), &Signature, &_MemContext));
        //
        PeakBytes = _Heap.Stats.NumInUseMax * sizeof(MPI_UNIT);
        //
        Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
        ++Loops;
    } while (Status >= 0 && Elapsed < Limit);
    //
Finally:
    CRYPTO_EdDSA_KillPrivateKey(&Private);
    CRYPTO_EdDSA_KillPublicKey (&Public);
    CRYPTO_EdDSA_KillSignature (&Signature);
    //
    if (Status < 0 || Loops == 0) {
        SEGGER_SYS_IO_Printf("%13s |", "-Fail-");
    } else if (_ShowMemory) {
        SEGGER_SYS_IO_Printf("%13d |", PeakBytes);
    } else {
        Loops *= 2; // Two agreements per loop
        Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
        if (_ShowMemory) {
            SEGGER_SYS_IO_Printf("%8d |", PeakBytes);
        } else {
            SEGGER_SYS_IO_Printf("%13.2f |", Time);
        }
    }
    SEGGER_SYS_IO_Printf("\n");
}

*****
*
*      Public code
*
*****
```

```
*****
*
*      MainTask()
*
*  Function description
*      Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
```

```

CRYPTO_Init();
SEGGER_SYS_Init();
SEGGER_MEM_SELFTEST_HEAP_Init(&_MemContext, &_Heap, _aUnits, MAX_CHUNKS, sizeof(MPI_UNIT));
//
SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
SEGGER_SYS_IO_Printf("EdDSA Sign and Verify Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
//
SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
if (SEGGER_SYS_GetProcessorSpeed() > 0) {
    SEGGER_SYS_IO_Printf("System: Processor speed      = %.3f MHz
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
}
SEGGER_SYS_IO_Printf("Config:  Static heap size      = %u bytes\n", sizeof(_aUnits));
SEGGER_SYS_IO_Printf("Config:  CRYPTO_VERSION      = %u
%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
SEGGER_SYS_IO_Printf("Config:  CRYPTO_MPI_BITS_PER_LIMB = %u\n", CRYPTO_MPI_BITS_PER_LIMB);
SEGGER_SYS_IO_Printf("\n");
//
SEGGER_SYS_IO_Printf("+-----+\n");
SEGGER_SYS_IO_Printf("| Curve      | ms/Sign      |\n");
SEGGER_SYS_IO_Printf("+-----+\n");
//_
_BenchmarkEd25519Sign();
_BenchmarkEd448Sign();
//
SEGGER_SYS_IO_Printf("+-----+\n");
SEGGER_SYS_IO_Printf("\n");
//
SEGGER_SYS_IO_Printf("+-----+\n");
SEGGER_SYS_IO_Printf("| Curve      | ms/Verify     |\n");
SEGGER_SYS_IO_Printf("+-----+\n");
//_
_BenchmarkEd25519Verify();
_BenchmarkEd448Verify();
//
SEGGER_SYS_IO_Printf("+-----+\n");
SEGGER_SYS_IO_Printf("\n");
//
SEGGER_SYS_IO_Printf("Benchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****/

```

Chapter 14

Key encapsulation

emCrypt implements the following key encapsulation methods:

- RSAES-OAEP
- RSAES-PKCS1

In addition to key encapsulation, emCrypt implements the following key wrapping methods:

- AES-KW
- SEED-KW
- ARIA-KW
- Camellia-KW
- Twofish-KW

14.1 RSAES-OAEP

14.1.1 Type-safe API

Function	Description
Encryption	
<code>CRYPTO_RSAES_OAEP_KDF1_SHA1_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA1.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA224_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA256_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA384_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA384.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA512.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA512/224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA512/256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA3-224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA3-256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA3-384.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Encrypt()</code>	Encrypt using RSA-OAEP-KDF1-SHA3-512.
Decryption	
<code>CRYPTO_RSAES_OAEP_KDF1_SHA1_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA1.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA224_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA256_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA384_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA384.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA512.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA512/224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA512/256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA3-224.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA3-256.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA3-384.
<code>CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Decrypt()</code>	Decrypt using RSA-OAEP-KDF1-SHA3-512.

14.1.1.1 CRYPTO_RSAES_OAEP_KDF1_SHA1_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA1.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA1_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                         U8                           * pOutput,
                                         unsigned                      OutputLen,
                                         const U8                      * pInput,
                                         unsigned                      InputLen,
                                         const U8                      * pLabel,
                                         unsigned                      LabelLen,
                                         CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.2 CRYPTO_RSAES_OAEP_KDF1_SHA224_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA224_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.3 CRYPTO_RSAES_OAEP_KDF1_SHA256_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA256_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.4 CRYPTO_RSAES_OAEP_KDF1_SHA384_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA384.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA384_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.5 CRYPTO_RSAES_OAEP_KDF1_SHA512_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA512.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.6 CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA512/224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.7 CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA512/256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.8 CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA3-224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.9 CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA3-256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.10 CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA3-384.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.11 CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Encrypt()

Description

Encrypt using RSA-OAEP-KDF1-SHA3-512.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Encrypt
    (const CRYPTO_RSA_PUBLIC_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string.
InputLen	Number of bytes in message.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- = 0 Failed, buffer is too small to hold encrypted message with OAEP padding.
- > 0 Number of bytes written to the output buffer.

Additional information

Encrypts the input plaintext to the output ciphertext using a public key and OAEP padding using a random mask generation seed.

The output buffer must be at least the octet length of the public key modulus.

14.1.1.12 CRYPTO_RSAES_OAEP_KDF1_SHA1_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA1.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA1_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                         U8                           * pOutput,
                                         unsigned                      OutputLen,
                                         const U8                      * pInput,
                                         unsigned                      InputLen,
                                         const U8                      * pLabel,
                                         unsigned                      LabelLen,
                                         CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
 ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.13 CRYPTO_RSAES_OAEP_KDF1_SHA224_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA224_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
 ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.14 CRYPTO_RSAES_OAEP_KDF1_SHA256_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA256_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
 ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.15 CRYPTO_RSAES_OAEP_KDF1_SHA384_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA384.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA384_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
 ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.16 CRYPTO_RSAES_OAEP_KDF1_SHA512_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA512.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_Decrypt(const CRYPTO_RSA_PRIVATE_KEY * pSelf,
                                            U8                           * pOutput,
                                            unsigned                      OutputLen,
                                            const U8                      * pInput,
                                            unsigned                      InputLen,
                                            const U8                      * pLabel,
                                            unsigned                      LabelLen,
                                            CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Processing error.
 ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.17 CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA512/224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.18 CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA512/256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.19 CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA3-224.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.20 CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA3-256.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.21 CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA3-384.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.1.22 CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Decrypt()

Description

Decrypt using RSA-OAEP-KDF1-SHA3-512.

Prototype

```
int CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Decrypt
    (const CRYPTO_RSA_PRIVATE_KEY * pSelf,
     U8                           * pOutput,
     unsigned                      OutputLen,
     const U8                      * pInput,
     unsigned                      InputLen,
     const U8                      * pLabel,
     unsigned                      LabelLen,
     CRYPTO_MEM_CONTEXT           * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to private key for decryption.
pOutput	Pointer to object that receives the decrypted message.
OutputLen	Octet length of the output object.
pInput	Pointer to message octet string to decrypt.
InputLen	Octet length of the message octet string.
pLabel	Pointer to label octet string.
LabelLen	Octet length of the label octet string.
pMem	Allocator to use for temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Number of bytes written to the output buffer.

Additional information

The output buffer must be at least the octet length of the private key's modulus.

14.1.2 Self-test API

The following table lists the RSAES-OAEP self-test API functions.

Function	Description
CRYPTO_RSAES_OAEP_EMU_SelfTest()	Run RSAES-OAEP test vectors from EMC.

14.1.2.1 CRYPTO_RSAES_OAEP_EMCSelfTest()

Description

Run RSAES-OAEP test vectors from EMC.

Prototype

```
void CRYPTO_RSAES_OAEP_EMCSelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator to use for temporary storage.

14.2 RSAES-PKCS1

14.2.1 Type-safe API

Function	Description
CRYPTO_RSAES_PKCS1_Encrypt()	Encrypt data using PKCS#1 version 1.5.

14.2.1.1 CRYPTO_RSAES_PKCS1_Encrypt()

Description

Encrypt data using PKCS#1 version 1.5.

Prototype

```
int CRYPTO_RSAES_PKCS1_Encrypt(const CRYPTO_RSA_PUBLIC_KEY * pSelf,
                                U8 * pOutput,
                                unsigned OutputLen,
                                const U8 * pInput,
                                unsigned InputLen,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to RSA public key to encrypt with.
pOutput	Pointer to object that receives the ciphertext.
OutputLen	Octet length of the ciphertext octet string.
pInput	Pointer to plaintext octet string.
InputLen	Octet length of the plaintext octet string.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Octet length of the ciphertext.

14.3 AES-KW

14.3.1 Standards reference

Key Wrap is specified by the following document:

- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*

The above specification standardizes the following documents:

- IETF RFC 3394 — *Advanced Encryption Standard (AES) Key Wrap Algorithm*
- IETF RFC 5649 — *Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*

14.3.2 Type-safe API

Function	Description
AESKW	
CRYPTO_KW_AESKW_Wrap()	Wrap key using AES.
CRYPTO_KW_AESKW_Unwrap()	Unwrap key using AES.
AESKWP	
CRYPTO_KW_AESKWP_Wrap()	Wrap key using AES, padded.
CRYPTO_KW_AESKWP_Unwrap()	Unwrap key using AES, padded.
NIST SP 800-38F primitives	
CRYPTO_KW_SP800_38F_AES_Wrap()	Wrap ICV and key using AES.
CRYPTO_KW_SP800_38F_AES_Unwrap()	Unwrap to ICV and key using AES.

14.3.2.1 CRYPTO_KW_AESKW_Wrap()

Description

Wrap key using AES.

Prototype

```
void CRYPTO_KW_AESKW_Wrap(      U8      * pOutput,
                                const U8      * pKey,
                                unsigned     KeyLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding AES-128, AES-192, and AES-256.

14.3.2.2 CRYPTO_KW_AESKW_Unwrap()

Description

Unwrap key using AES.

Prototype

```
int CRYPTO_KW_AESKW_Unwrap(      U8      * pOutput,
                                const U8      * pInput,
                                unsigned     InputLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the unwrapped key.
pInput	Pointer to object that contains the wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
≥ 0 Octet length of unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding AES-128, AES-192, and AES-256.

14.3.2.3 CRYPTO_KW_AESKWP_Wrap()

Description

Wrap key using AES, padded.

Prototype

```
void CRYPTO_KW_AESKWP_Wrap(      U8      * pOutput,
                                const U8      * pKey,
                                unsigned     KeyLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding AES-128, AES-192, and AES-256.

14.3.2.4 CRYPTO_KW_AESKWP_Unwrap()

Description

Unwrap key using AES, padded.

Prototype

```
int CRYPTO_KW_AESKWP_Unwrap(      U8      * pKey,
                                const U8      * pInput,
                                unsigned     InputLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
≥ 0 Octet length of the unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding AES-128, AES-192, and AES-256.

14.3.2.5 CRYPTO_KW_SP800_38F_AES_Wrap()

Description

Wrap ICV and key using AES.

Prototype

```
void CRYPTO_KW_SP800_38F_AES_Wrap(      U8      * pOutput,
                                         const U8      * pICV,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pKEK,
                                         unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pICV	Pointer to integrity check value.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding AES-128, AES-192, and AES-256.

14.3.2.6 CRYPTO_KW_SP800_38F_AES_Unwrap()

Description

Unwrap to ICV and key using AES.

Prototype

```
void CRYPTO_KW_SP800_38F_AES_Unwrap(      U8      * pICV,
                                            U8      * pKey,
                                            const U8      * pInput,
                                            unsigned     InputLen,
                                            const U8      * pKEK,
                                            unsigned     KEKLen);
```

Parameters

Parameter	Description
pICV	Pointer to object that receives the ICV.
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key. The KEK sizes supported are 16, 24, and 32 bytes corresponding to AES-128, AES-192, and AES-256.
KEKLen	Octet length of the key encryption key.

Additional information

The first 8 bytes of the unwrapped data are the ICV which, for NIST compliance, should all be 0xA6 (assuming this ICV for wrapping).

Other key wrapping schemes, such as X9.102's AESKW, specify a different ICV. It is the client's responsibility to check the IV to ensure the key material is correctly recovered after unwrapping.

14.3.3 Self-test API

The following table lists the AESKW self-test API functions.

Function	Description
CRYPTO_AESKW_RFC3394_SelfTest()	Run AESKW KATs from RFC 3394.

14.3.3.1 CRYPTO_AESKW_RFC3394_SelfTest()

Description

Run AESKW KATs from RFC 3394.

Prototype

```
void CRYPTO_AESKW_RFC3394_SelfTest(const CRYPTO_SELFTEST_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.

14.4 SEED-KW

14.4.1 Standards reference

Key Wrap is specified by the following document:

- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*

14.4.2 Type-safe API

Function	Description
SEEDKW	
<code>CRYPTO_KW_SEEDKW_Wrap()</code>	Wrap key using SEED.
<code>CRYPTO_KW_SEEDKW_Unwrap()</code>	Unwrap key using SEED.
SEEDKWP	
<code>CRYPTO_KW_SEEDKWP_Wrap()</code>	Wrap key using SEED, padded.
<code>CRYPTO_KW_SEEDKWP_Unwrap()</code>	Unwrap key using SEED, padded.
NIST SP 800-38F primitives	
<code>CRYPTO_KW_SP800_38F_SEED_Wrap()</code>	Wrap ICV and key using SEED.
<code>CRYPTO_KW_SP800_38F_SEED_Unwrap()</code>	Unwrap to ICV and key using SEED.

14.4.2.1 CRYPTO_KW_SEEDKW_Wrap()

Description

Wrap key using SEED.

Prototype

```
void CRYPTO_KW_SEEDKW_Wrap(      U8      * pOutput,
                                  const U8      * pKey,
                                  unsigned     KeyLen,
                                  const U8      * pKEK,
                                  unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding SEED-128, SEED-192, and SEED-256.

14.4.2.2 CRYPTO_KW_SEEDKW_Unwrap()

Description

Unwrap key using SEED.

Prototype

```
int CRYPTO_KW_SEEDKW_Unwrap(      U8      * pOutput,
                                const U8      * pInput,
                                unsigned     InputLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the unwrapped key.
pInput	Pointer to object that contains the wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding SEED-128, SEED-192, and SEED-256.

14.4.2.3 CRYPTO_KW_SEEDKWP_Wrap()

Description

Wrap key using SEED, padded.

Prototype

```
void CRYPTO_KW_SEEDKWP_Wrap(      U8      * pOutput,
                                    const U8      * pKey,
                                    unsigned      KeyLen,
                                    const U8      * pKEK,
                                    unsigned      KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding SEED-128, SEED-192, and SEED-256.

14.4.2.4 CRYPTO_KW_SEEDKWP_Unwrap()

Description

Unwrap key using SEED, padded.

Prototype

```
int CRYPTO_KW_SEEDKWP_Unwrap(      U8      * pKey,
                                  const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8      * pKEK,
                                  unsigned     KEKLen);
```

Parameters

Parameter	Description
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of the unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding SEED-128, SEED-192, and SEED-256.

14.4.2.5 CRYPTO_KW_SP800_38F_SEED_Wrap()

Description

Wrap ICV and key using SEED.

Prototype

```
void CRYPTO_KW_SP800_38F_SEED_Wrap(      U8      * pOutput,
                                         const U8      * pICV,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pKEK,
                                         unsigned      KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pICV	Pointer to integrity check value.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding SEED-128, SEED-192, and SEED-256.

14.4.2.6 CRYPTO_KW_SP800_38F_SEED_Unwrap()

Description

Unwrap to ICV and key using SEED.

Prototype

```
void CRYPTO_KW_SP800_38F_SEED_Unwrap(      U8      * pICV,
                                             U8      * pKey,
                                             const U8      * pInput,
                                             unsigned   InputLen,
                                             const U8      * pKEK,
                                             unsigned   KEKLen);
```

Parameters

Parameter	Description
pICV	Pointer to object that receives the ICV.
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key. The KEK sizes supported are 16, 24, and 32 bytes corresponding to SEED-128, SEED-192, and SEED-256.
KEKLen	Octet length of the key encryption key.

Additional information

The first 8 bytes of the unwrapped data are the ICV which, for NIST compliance, should all be 0xA6 (assuming this ICV for wrapping).

Other key wrapping schemes, such as X9.102's AESKW, specify a different ICV. It is the client's responsibility to check the IV to ensure the key material is correctly recovered after unwrapping.

14.5 ARIA-KW

14.5.1 Standards reference

Key Wrap is specified by the following document:

- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*

14.5.2 Type-safe API

Function	Description
ARIAKW	
CRYPTO_KW_ARIAKW_Wrap()	Wrap key using ARIA.
CRYPTO_KW_ARIAKW_Unwrap()	Unwrap key using ARIA.
ARIAKWP	
CRYPTO_KW_ARIAKWP_Wrap()	Wrap key using ARIA, padded.
CRYPTO_KW_ARIAKWP_Unwrap()	Unwrap key using ARIA, padded.
NIST SP 800-38F primitives	
CRYPTO_KW_SP800_38F_ARIA_Wrap()	Wrap ICV and key using ARIA.
CRYPTO_KW_SP800_38F_ARIA_Unwrap()	Unwrap to ICV and key using ARIA.

14.5.2.1 CRYPTO_KW_ARIAKW_Wrap()

Description

Wrap key using ARIA.

Prototype

```
void CRYPTO_KW_ARIAKW_Wrap(      U8      * pOutput,
                                  const U8      * pKey,
                                  unsigned     KeyLen,
                                  const U8      * pKEK,
                                  unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding ARIA-128, ARIA-192, and ARIA-256.

14.5.2.2 CRYPTO_KW_ARIAKW_Unwrap()

Description

Unwrap key using ARIA.

Prototype

```
int CRYPTO_KW_ARIAKW_Unwrap(      U8      * pOutput,
                                const U8      * pInput,
                                unsigned     InputLen,
                                const U8      * pKEK,
                                unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the unwrapped key.
pInput	Pointer to object that contains the wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding ARIA-128, ARIA-192, and ARIA-256.

14.5.2.3 CRYPTO_KW_ARIAKWP_Wrap()

Description

Wrap key using ARIA, padded.

Prototype

```
void CRYPTO_KW_ARIAKWP_Wrap(      U8      * pOutput,
                                    const U8      * pKey,
                                    unsigned      KeyLen,
                                    const U8      * pKEK,
                                    unsigned      KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding ARIA-128, ARIA-192, and ARIA-256.

14.5.2.4 CRYPTO_KW_ARIAKWP_Unwrap()

Description

Unwrap key using ARIA, padded.

Prototype

```
int CRYPTO_KW_ARIAKWP_Unwrap(      U8      * pKey,
                                  const U8      * pInput,
                                  unsigned     InputLen,
                                  const U8      * pKEK,
                                  unsigned     KEKLen);
```

Parameters

Parameter	Description
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of the unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding ARIA-128, ARIA-192, and ARIA-256.

14.5.2.5 CRYPTO_KW_SP800_38F_ARIA_Wrap()

Description

Wrap ICV and key using ARIA.

Prototype

```
void CRYPTO_KW_SP800_38F_ARIA_Wrap(      U8      * pOutput,
                                         const U8      * pICV,
                                         const U8      * pKey,
                                         unsigned      KeyLen,
                                         const U8      * pKEK,
                                         unsigned      KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pICV	Pointer to integrity check value.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding ARIA-128, ARIA-192, and ARIA-256.

14.5.2.6 CRYPTO_KW_SP800_38F_ARIA_Unwrap()

Description

Unwrap to ICV and key using ARIA.

Prototype

```
void CRYPTO_KW_SP800_38F_ARIA_Unwrap(      U8      * pICV,
                                            U8      * pKey,
                                            const U8      * pInput,
                                            unsigned   InputLen,
                                            const U8      * pKEK,
                                            unsigned   KEKLen);
```

Parameters

Parameter	Description
pICV	Pointer to object that receives the ICV.
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key. The KEK sizes supported are 16, 24, and 32 bytes corresponding to ARIA-128, ARIA-192, and ARIA-256.
KEKLen	Octet length of the key encryption key.

Additional information

The first 8 bytes of the unwrapped data are the ICV which, for NIST compliance, should all be 0xA6 (assuming this ICV for wrapping).

Other key wrapping schemes, such as X9.102's AESKW, specify a different ICV. It is the client's responsibility to check the IV to ensure the key material is correctly recovered after unwrapping.

14.6 Camellia-KW

14.6.1 Standards reference

Key Wrap is specified by the following document:

- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*

14.6.2 Type-safe API

Function	Description
CAMELLIAKW	
CRYPTO_KW_CAMELLIAKW_Wrap()	Wrap key using Camellia.
CRYPTO_KW_CAMELLIAKW_Unwrap()	Unwrap key using Camellia.
CAMELLIAKWP	
CRYPTO_KW_CAMELLIAKWP_Wrap()	Wrap key using Camellia, padded.
CRYPTO_KW_CAMELLIAKWP_Unwrap()	Unwrap key using Camellia, padded.
NIST SP 800-38F primitives	
CRYPTO_KW_SP800_38F_CAMELLIA_Wrap()	Wrap ICV and key using Camellia.
CRYPTO_KW_SP800_38F_CAMELLIA_Unwrap()	Unwrap to ICV and key using Camellia.

14.6.2.1 CRYPTO_KW_CAMELLIAKW_Wrap()

Description

Wrap key using Camellia.

Prototype

```
void CRYPTO_KW_CAMELLIAKW_Wrap(      U8      * pOutput ,
                                         const U8      * pKey ,
                                         unsigned      KeyLen ,
                                         const U8      * pKEK ,
                                         unsigned      KEKLen );
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Camellia-128, Camellia-192, and Camellia-256.

14.6.2.2 CRYPTO_KW_CAMELLIAKW_Unwrap()

Description

Unwrap key using Camellia.

Prototype

```
int CRYPTO_KW_CAMELLIAKW_Unwrap(      U8      * pOutput,
                                       const U8      * pInput,
                                       unsigned   InputLen,
                                       const U8      * pKEK,
                                       unsigned   KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the unwrapped key.
pInput	Pointer to object that contains the wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding Camellia-128, Camellia-192, and Camellia-256.

14.6.2.3 CRYPTO_KW_CAMELLIAKWP_Wrap()

Description

Wrap key using Camellia, padded.

Prototype

```
void CRYPTO_KW_CAMELLIAKWP_Wrap(      U8      * pOutput,
                                         const U8      * pKey,
                                         unsigned     KeyLen,
                                         const U8      * pKEK,
                                         unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Camellia-128, Camellia-192, and Camellia-256.

14.6.2.4 CRYPTO_KW_CAMELLIAKWP_Unwrap()

Description

Unwrap key using Camellia, padded.

Prototype

```
int CRYPTO_KW_CAMELLIAKWP_Unwrap(      U8      * pKey,
                                         const U8      * pInput,
                                         unsigned   InputLen,
                                         const U8      * pKEK,
                                         unsigned   KEKLen);
```

Parameters

Parameter	Description
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of the unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding Camellia-128, Camellia-192, and Camellia-256.

14.6.2.5 CRYPTO_KW_SP800_38F_CAMELLIA_Wrap()

Description

Wrap ICV and key using Camellia.

Prototype

```
void CRYPTO_KW_SP800_38F_CAMELLIA_Wrap(      U8      * pOutput,
                                              const U8    * pICV,
                                              const U8    * pKey,
                                              unsigned   KeyLen,
                                              const U8    * pKEK,
                                              unsigned   KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pICV	Pointer to integrity check value.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Camellia-128, Camellia-192, and Camellia-256.

14.6.2.6 CRYPTO_KW_SP800_38F_CAMELLIA_Unwrap()

Description

Unwrap to ICV and key using Camellia.

Prototype

```
void CRYPTO_KW_SP800_38F_CAMELLIA_Unwrap(      U8      * pICV,
                                              U8      * pKey,
                                              const U8      * pInput,
                                              unsigned   InputLen,
                                              const U8      * pKEK,
                                              unsigned   KEKLen);
```

Parameters

Parameter	Description
pICV	Pointer to object that receives the ICV.
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key. The KEK sizes supported are 16, 24, and 32 bytes corresponding to Camellia-128, Camellia-192, and Camellia-256.
KEKLen	Octet length of the key encryption key.

Additional information

The first 8 bytes of the unwrapped data are the ICV which, for NIST compliance, should all be 0xA6 (assuming this ICV for wrapping).

Other key wrapping schemes, such as X9.102's AESKW, specify a different ICV. It is the client's responsibility to check the IV to ensure the key material is correctly recovered after unwrapping.

14.7 Twofish-KW

14.7.1 Standards reference

Key Wrap is specified by the following document:

- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*

14.7.2 Type-safe API

Function	Description
TWOFISHKW	
CRYPTO_KW_TWOFOISHKW_Wrap()	Wrap key using Twofish.
CRYPTO_KW_TWOFOISHKW_Unwrap()	Unwrap key using Twofish.
TWOFISHKWP	
CRYPTO_KW_TWOFOISHKWP_Wrap()	Wrap key using Twofish, padded.
CRYPTO_KW_TWOFOISHKWP_Unwrap()	Unwrap key using Twofish, padded.
NIST SP 800-38F primitives	
CRYPTO_KW_SP800_38F_TWOFOISH_Wrap()	Wrap ICV and key using Twofish.
CRYPTO_KW_SP800_38F_TWOFOISH_Unwrap()	Unwrap to ICV and key using Twofish.

14.7.2.1 CRYPTO_KW_TWOFISHKW_Wrap()

Description

Wrap key using Twofish.

Prototype

```
void CRYPTO_KW_TWOFISHKW_Wrap(      U8      * pOutput,
                                     const U8      * pKey,
                                     unsigned     KeyLen,
                                     const U8      * pKEK,
                                     unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Twofish-128, Twofish-192, and Twofish-256.

14.7.2.2 CRYPTO_KW_TWOFISHKW_Unwrap()

Description

Unwrap key using Twofish.

Prototype

```
int CRYPTO_KW_TWOFISHKW_Unwrap(      U8      * pOutput,
                                    const U8      * pInput,
                                    unsigned     InputLen,
                                    const U8      * pKEK,
                                    unsigned     KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the unwrapped key.
pInput	Pointer to object that contains the wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding Twofish-128, Twofish-192, and Twofish-256.

14.7.2.3 CRYPTO_KW_TWOFISHKWP_Wrap()

Description

Wrap key using Twofish, padded.

Prototype

```
void CRYPTO_KW_TWOFISHKWP_Wrap(      U8      * pOutput ,
                                         const U8      * pKey ,
                                         unsigned      KeyLen ,
                                         const U8      * pKEK ,
                                         unsigned      KEKLen );
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Twofish-128, Twofish-192, and Twofish-256.

14.7.2.4 CRYPTO_KW_TWOFISHKWP_Unwrap()

Description

Unwrap key using Twofish, padded.

Prototype

```
int CRYPTO_KW_TWOFISHKWP_Unwrap(      U8      * pKey,
                                      const U8      * pInput,
                                      unsigned   InputLen,
                                      const U8      * pKEK,
                                      unsigned   KEKLen);
```

Parameters

Parameter	Description
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Return value

< 0 Error unwrapping key.
 ≥ 0 Octet length of the unwrapped key.

Additional information

When the key cannot be successfully unwrapped, all key data is erased.

The KEK sizes supported are 16, 24, and 32 bytes corresponding Twofish-128, Twofish-192, and Twofish-256.

14.7.2.5 CRYPTO_KW_SP800_38F_TWOFISH_Wrap()

Description

Wrap ICV and key using Twofish.

Prototype

```
void CRYPTO_KW_SP800_38F_TWOFISH_Wrap(      U8          * pOutput,
                                             const U8        * pICV,
                                             const U8        * pKey,
                                             unsigned       KeyLen,
                                             const U8        * pKEK,
                                             unsigned       KEKLen);
```

Parameters

Parameter	Description
pOutput	Pointer to object that receives the wrapped ICV and key.
pICV	Pointer to integrity check value.
pKey	Pointer to key to be wrapped.
KeyLen	Octet length of the key.
pKEK	Pointer to key encryption key.
KEKLen	Octet length of key encryption key.

Additional information

The KEK sizes supported are 16, 24, and 32 bytes corresponding Twofish-128, Twofish-192, and Twofish-256.

14.7.2.6 CRYPTO_KW_SP800_38F_TWOFISH_Unwrap()

Description

Unwrap to ICV and key using Twofish.

Prototype

```
void CRYPTO_KW_SP800_38F_TWOFISH_Unwrap(      U8      * pICV,
                                              U8      * pKey,
                                              const U8      * pInput,
                                              unsigned   InputLen,
                                              const U8      * pKEK,
                                              unsigned   KEKLen);
```

Parameters

Parameter	Description
pICV	Pointer to object that receives the ICV.
pKey	Pointer to object that receives the unwrapped key.
pInput	Pointer to wrapped ICV and key.
InputLen	Octet length of the wrapped ICV and key.
pKEK	Pointer to key encryption key. The KEK sizes supported are 16, 24, and 32 bytes corresponding to Twofish-128, Twofish-192, and Twofish-256.
KEKLen	Octet length of the key encryption key.

Additional information

The first 8 bytes of the unwrapped data are the ICV which, for NIST compliance, should all be 0xA6 (assuming this ICV for wrapping).

Other key wrapping schemes, such as X9.102's AESKW, specify a different ICV. It is the client's responsibility to check the IV to ensure the key material is correctly recovered after unwrapping.

Chapter 15

Key agreement

emCrypt implements the following key agreement methods:

- Diffie-Hellman key agreement
- Elliptic curve Diffie-Hellman key agreement

15.1 Overview

A key agreement protocol (or key exchange protocol), is a sequence of steps used by two or more parties when they need to agree upon a single key to use for a secret-key cryptosystem. Such protocols enable users to share keys, securely, over any insecure medium, and to do so without a previously-established shared secret.

15.2 Diffie-Hellman key agreement

15.2.1 Data types

Type	Description
CRYPTO_DH_KA_CONTEXT	ECDH key agreement data.

15.2.1.1 CRYPTO_DH_KA_CONTEXT

Description

ECDH key agreement data.

Type definition

```
typedef struct {
    CRYPTO_MPI P;
    CRYPTO_MPI G;
    CRYPTO_MPI X;
    CRYPTO_MPI Y;
    CRYPTO_MPI K;
} CRYPTO_DH_KA_CONTEXT;
```

Structure members

Member	Description
P	Field modulus
G	Generator
X	Secret value X
Y	Public value Y, G^X
K	Agreed key

15.2.2 Type-safe API

The following table lists the DH key agreement functions.

Function	Description
CRYPTO_DH_KA_Init()	Initialize DH key agreement context.
CRYPTO_DH_KA_Start()	Start DH key agreement protocol.
CRYPTO_DH_KA_Agree()	Generate shared secret.
CRYPTO_DH_KA_Kill()	Destroy DH key agreement context.
CRYPTO_DH_KA_GenKeys()	Generate ephemeral keys.

15.2.2.1 CRYPTO_DH_KA_Init()

Description

Initialize DH key agreement context.

Prototype

```
void CRYPTO_DH_KA_Init(CRYPTO_DH_KA_CONTEXT * pSelf,  
                        CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.
pMem	Allocator to use for temporary storage.

15.2.2.2 CRYPTO_DH_KA_Start()

Description

Start DH key agreement protocol.

Prototype

```
int CRYPTO_DH_KA_Start(      CRYPTO_DH_KA_CONTEXT * pSelf,
                           const CRYPTO_MPI        * pGenerator,
                           const CRYPTO_MPI        * pModulus,
                           CRYPTO_MEM_CONTEXT     * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.
pGenerator	Pointer to DH group generator.
pModulus	Pointer to DH group modulus.
pMem	Allocator to use for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

15.2.2.3 CRYPTO_DH_KA_Agree()

Description

Generate shared secret.

Prototype

```
int CRYPTO_DH_KA_Agree( CRYPTO_DH_KA_CONTEXT * pSelf,
                         const CRYPTO_MPI          * pPeer,
                         CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.
pPeer	Pointer to peer's public key.
pMem	Allocator to use for temporary storage.

Return value

≥ 0 Success.
 < 0 Processing error.

Additional information

Calculates the shared secret $(G^Y)^X \bmod P$.

15.2.2.4 CRYPTO_DH_KA_Kill()

Description

Destroy DH key agreement context.

Prototype

```
void CRYPTO_DH_KA_Kill(CRYPTO_DH_KA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.

15.2.2.5 CRYPTO_DH_KA_GenKeys()

Description

Generate ephemeral keys.

Prototype

```
int CRYPTO_DH_KA_GenKeys(CRYPTO_DH_KA_CONTEXT * pSelf,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.
pMem	Allocator to use for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

This function chooses a random private value, X, and computes the corresponding public value Y which is G^X . Note that the key agreement context must be assigned a valid prime and generator before entry.

15.2.3 Self-test API

The following table lists the DH key agreement self-test API functions.

Function	Description
CRYPTO_DH_KA_SEGGER_SelfTest()	Run DH-KA self tests from SEGGER.

15.2.3.1 CRYPTO_DH_KA_SEGGER_SelfTest()

Description

Run DH-KA self tests from SEGGER.

Prototype

```
void CRYPTO_DH_KA_SEGGER_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Allocator to use for temporary storage.

Additional information

Tests are run over all DH groups.

15.3 Elliptic curve Diffie-Hellman key agreement

15.3.1 Data types

Type	Description
CRYPTO_ECDH_KA_CONTEXT	ECDH key agreement data.

15.3.1.1 CRYPTO_ECDH_KA_CONTEXT

Description

ECDH key agreement data.

Type definition

```
typedef struct {
    CRYPTO_ECDSA_PUBLIC_KEY    Public;
    CRYPTO_ECDSA_PRIVATE_KEY   Private;
    CRYPTO_EC_POINT            PeerPublic;
    CRYPTO_EC_POINT            K;
} CRYPTO_ECDH_KA_CONTEXT;
```

Structure members

Member	Description
Public	Our public key
Private	Our private key
PeerPublic	Peer's public key, curve is implicit.
K	Agreed key; the X coordinate is all we require.

15.3.2 Type-safe API

The following table lists the ECDH key agreement functions.

Function	Description
CRYPTO_ECDH_KA_Init()	Initialize ECDH key agreement context.
CRYPTO_ECDH_KA_Start()	Start ECDH key agreement protocol.
CRYPTO_ECDH_KA_StartEx()	Start ECDH key agreement protocol, specify private key.
CRYPTO_ECDH_KA_Agree()	Generate shared secret.
CRYPTO_ECDH_KA_Kill()	Destroy ECDH key agreement context.

15.3.2.1 CRYPTO_ECDH_KA_Init()

Description

Initialize ECDH key agreement context.

Prototype

```
void CRYPTO_ECDH_KA_Init(CRYPTO_ECDH_KA_CONTEXT * pSelf,  
                          CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to ECDH key agreement context.
pMem	Allocator to use for temporary storage.

15.3.2.2 CRYPTO_ECDH_KA_Start()

Description

Start ECDH key agreement protocol.

Prototype

```
int CRYPTO_ECDH_KA_Start(      CRYPTO_ECDH_KA_CONTEXT * pSelf,
                               const CRYPTO_EC_CURVE    * pCurve,
                               CRYPTO_MEM_CONTEXT       * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to ECDH key agreement context.
pCurve	Pointer to elliptic curve.
pMem	Allocator to use for temporary storage.

Return value

≥ 0 Success.
< 0 Processing error.

15.3.2.3 CRYPTO_ECDH_KA_StartEx()

Description

Start ECDH key agreement protocol, specify private key.

Prototype

```
int CRYPTO_ECDH_KA_StartEx( CRYPTO_ECDH_KA_CONTEXT * pSelf,
                           const CRYPTO_MPI           * pD,
                           const CRYPTO_EC_CURVE       * pCurve,
                           CRYPTO_MEM_CONTEXT          * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to ECDH key agreement context.
pD	Pointer to EC secret key.
pCurve	Pointer to elliptic curve.
pMem	Allocator to use for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

15.3.2.4 CRYPTO_ECDH_KA_Agree()

Description

Generate shared secret.

Prototype

```
int CRYPTO_ECDH_KA_Agree( CRYPTO_ECDH_KA_CONTEXT * pSelf,  
                           const CRYPTO_EC_POINT      * pPeer,  
                           CRYPTO_MEM_CONTEXT        * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to DH key agreement context.
pPeer	Pointer to peer's public key.
pMem	Allocator to use for temporary storage.

Return value

≥ 0 Success.
< 0 Processing error.

Additional information

Calculates the shared secret.

15.3.2.5 CRYPTO_ECDH_KA_Kill()

Description

Destroy ECDH key agreement context.

Prototype

```
void CRYPTO_ECDH_KA_Kill(CRYPTO_ECDH_KA_CONTEXT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to ECDH key agreement context.

15.3.3 Self-test API

The following table lists the ECDH key agreement self-test API functions.

Function	Description
CRYPTO_ECDH_KA SEGGER_SelfTest()	Run DH self-test over all IETF DH groups.

15.3.3.1 CRYPTO_ECDH_KA_SEGGER_SelfTest()

Description

Run DH self-test over all IETF DH groups.

Prototype

```
void CRYPTO_ECDH_KA_SEGGER_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Allocator to use for temporary storage.

15.3.4 Example applications

15.3.4.1 CRYPTO_Bench_ECDH.c

This application benchmarks the configured performance of ECDH key agreement.

Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
ECDH Key Agreement Benchmark compiled Mar 19 2018 16:31:17
```

```
Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: Static heap size         = 3256 bytes
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_MPI_BITS_PER_LIMB = 32
```

This benchmarks both ends of an ECDH key agreement, but.
timing is reported as the time for one end's calculation.

Curve	ms/Agreement	Memory
secp192r1	92.12	704
secp192k1	126.01	704
secp224r1	103.09	792
secp224k1	165.02	792
secp256r1	151.01	880
secp256k1	206.17	880
secp384r1	268.57	1232
secp512r1	467.70	1628
brainpoolP160r1	100.40	616
brainpoolP160t1	90.84	616
brainpoolP192r1	131.17	704
brainpoolP192t1	122.63	704
brainpoolP224r1	173.96	792
brainpoolP224t1	158.12	792
brainpoolP256r1	225.84	880
brainpoolP256t1	209.92	880
brainpoolP320r1	340.74	1056
brainpoolP320t1	313.00	1056
brainpoolP384r1	538.10	1232
brainpoolP384t1	486.32	1232
brainpoolP512r1	969.11	1584
brainpoolP512t1	882.77	1584

Benchmark complete

Complete listing

```
/*
 *      (c) SEGGER Microcontroller GmbH
 *      The Embedded Experts
 *      www.segger.com
 ****

----- END-OF-HEADER -----

File      : CRYPTO_Bench_ECDH.c
Purpose   : Benchmark ECDH key agreement performance.

*/
/*
 *      #include section
 *
 ****
```

```
/*
#include "CRYPTO.h"
#include "SEGGER_MEM.h"
#include "SEGGER_SYS.h"

***** Defines, configurable *****
*/
*      Defines, configurable
*
***** Defines, fixed *****
*/
#define MAX_CHUNKS          22

***** Local data types *****
*/
*      Local data types
*
***** Static data *****
*/
static MPI_UNIT           _aUnits[MAX_CHUNKS];
static SEGGER_MEM_CONTEXT _MemContext;
static SEGGER_MEM_SELFTEST_HEAP _Heap;

***** Static code *****
*/
*      _LFSR_Get()
*
* Function description
*   Get pseudo-random data.
*
* Parameters
*   pData - Pointer to object the receives the random data.
*   DataLen - Octet length of the random data.
*
* Additional information
*   This LFSR PRNG does not need to be cryptographically strong as
*   its purpose is only to deliver repeatable pseudo-random data
*   that does not depend upon a nondeterministic source (such as
*   a hardware RNG or the availability of hardware ciphering and
*   hashing in the DRBG code). Therefore, this RNG is suitable
*   for deterministic benchmarking across compilers and systems.
*/
static void _LFSR_Get(U8 *pData, unsigned DataLen) {
    static U32 State32 = 0xFEDCBA8uL;
    static U32 State31 = 0x1234567uL;
    //
    while (DataLen > 0) {
        if (State32 & 1) {
            State32 >>= 1;
            State32 ^= 0xB4BCD35CuL;
        } else {
            State32 >>= 1;
        }
        if (State32 & 1) {
            State31 >>= 1;
        }
    }
}
```

```

        State31 ^= 0x7A5BC2E3uL;
    } else {
        State31 >>= 1;
    }
    if (DataLen >= 2) {
        CRYPTO_WRU16LE(pData, (U16)(State31 ^ State32));
        pData += 2;
        DataLen -= 2;
    } else {
        *pData = (U8)(State31 ^ State32);
        DataLen -= 1;
    }
}
}

//*****************************************************************************
*
*     _ConvertTicksToSeconds()
*
* Function description
*   Convert ticks to seconds.
*
* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static float _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0f;
}

//*****************************************************************************
*
*     _BenchmarkECDHKeyAgreement()
*
* Function description
*   Benchmark ECDH key agreement, both sides.
*
* Parameters
*   pCurve - Pointer to elliptic curve.
*/
static void _BenchmarkECDHKeyAgreement(const CRYPTO_EC_CURVE *pCurve) {
    CRYPTO_ECDH_KA_CONTEXT Us;
    CRYPTO_ECDH_KA_CONTEXT Them;
    U64 OneSecond;
    U64 T0;
    U64 Elapsed;
    int Loops;
    int Status;
    unsigned PeakBytes;
    unsigned UnitSize;
    float Time;
    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops = 0;
    Elapsed = 0;
    PeakBytes = 0;
    UnitSize = CRYPTO_MPI_BYTES_REQUIRED(2*CRYPTO_MPI_BitCount(&pCurve->p)+63) + 2*CRYPTO_MPI_BYTES_PER_LIMB;
    //
    SEGGER_SYS_IO_Printf(" | %-16s | ", pCurve->aCurveName);
    //
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        //
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        //
        CRYPTO_ECDH_KA_Init(&Us, &_MemContext);
        CRYPTO_ECDH_KA_Init(&Them, &_MemContext);
        //
        CRYPTO_CHECK(CRYPTO_ECDH_KA_Start(&Us, pCurve, &_MemContext));
        CRYPTO_CHECK(CRYPTO_ECDH_KA_Start(&Them, pCurve, &_MemContext));
        //
        CRYPTO_CHECK(CRYPTO_ECDH_KA_Agree(&Us, &Them.Public.Y, &_MemContext));
        CRYPTO_CHECK(CRYPTO_ECDH_KA_Agree(&Them, &Us.Public.Y, &_MemContext));
        //
        CRYPTO_ASSERT(CRYPTO_MPI_IsEqual(&Us.K.X, &Them.K.X));
        //
        CRYPTO_ECDH_KA_Kill(&Us);
        CRYPTO_ECDH_KA_Kill(&Them);
        //
        PeakBytes = _Heap.Stats.NumInUseMax * UnitSize;
        //
        Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
    }
}

```

```

    ++Loops;
} while (Status >= 0 && Elapsed < OneSecond);
//  

Finally:  

CRYPTO_ECDH_KA_Kill(&Us);  

CRYPTO_ECDH_KA_Kill(&Them);  

//  

if (Status < 0 || Loops == 0) {  

    SEGGER_SYS_IO_Printf("%13s |%13s | ", "-Fail-", "-Fail-");  

} else {  

    Loops *= 2; // Two agreements per loop  

    PeakBytes /= 2; // Two agreements per loop  

    Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;  

    SEGGER_SYS_IO_Printf("%13.2f |%13d | ", Time, PeakBytes);  

}  

SEGGER_SYS_IO_Printf("\n");
}  

*****  

*  

*      Public code  

*  

*****  

*/  

*****  

*  

*      MainTask()  

*  

*  Function description  

*  Main entry point for application to run all the tests.  

*/  

void MainTask(void);  

void MainTask(void) {
    static const CRYPTO_RNG_API LFSR = { NULL, _LFSR_Get, NULL, NULL };
    //  

    CRYPTO_Init();  

    CRYPTO_RNG_Install(&LFSR);  

    SEGGER_SYS_Init();  

    SEGGER_MEM_SELFTEST_HEAP_Init(&_MemContext, &_Heap, _aUnits, MAX_CHUNKS, sizeof(MPI_UNIT));
    //  

    SEGGER_SYS_IO_Printf("%s     www.segger.com\n", CRYPTO_GetCopyrightText());  

    SEGGER_SYS_IO_Printf("ECDH Key Agreement Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //  

    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());  

    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   Static heap size       = %u bytes\n", sizeof(_aUnits));
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION          = %u
%s\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_MPI_BITS_PER_LIMB = %u\n", CRYPTO_MPI_BITS_PER_LIMB);
    SEGGER_SYS_IO_Printf("\n");
    //  

    SEGGER_SYS_IO_Printf("This benchmarks both ends of an ECDH key agreement, but\n");
    SEGGER_SYS_IO_Printf("timing is reported as the time for one end's calculation.\n");
    SEGGER_SYS_IO_Printf("\n");
    SEGGER_SYS_IO_Printf("+-----+-----+-----+\n");
    SEGGER_SYS_IO_Printf(" | Curve           | ms/Agreement | Memory |\n");
    SEGGER_SYS_IO_Printf("+-----+-----+-----+\n");
    //  

    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp192r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp192k1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp224r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp224k1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp256r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp256k1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp384r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_secp512r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP160r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP160t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP192r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP192t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP224r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP224t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP256r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP256t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP320r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP320t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP384r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP384t1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP512r1);
    _BenchmarkECDHKeyAgreement(&CRYPTO_EC_CURVE_briapoolP512t1);
    //  

    SEGGER_SYS_IO_Printf("-----+\n");
    SEGGER_SYS_IO_Printf("\n");
}

```

```
//  
SEGGER_SYS_IO_Printf("Benchmark complete\n");  
SEGGER_SYS_OS_PauseBeforeHalt();  
SEGGER_SYS_OS_Halt(0);  
}  
  
***** End of file *****
```

Chapter 16

Elliptic curves

16.1 Overview

The following table compares the key sizes for RSA and elliptic curve cryptosystems for a given security level.

Security level	RSA key length	ECC key length	Ratio
80	1024	160--223	5--6 : 1
112	2048	224--255	8--9 : 1
128	3072	256--283	11--12 : 1
192	7680	384--511	15--20 : 1
256	15360	512--571	27--30 : 1

16.2 Data types

Type	Description
CRYPTO_EC_CURVE	Elliptic curve.
CRYPTO_EC_POINT	Elliptic curve point.

16.2.1 CRYPTO_EC_CURVE

Description

Elliptic curve.

Type definition

```
typedef struct {
    CRYPTO_MPI          P;
    CRYPTO_MPI          A;
    CRYPTO_MPI          B;
    CRYPTO_EC_POINT     G;
    CRYPTO_MPI          Q;
    U8                  OptimizedA;
    char                aCurveName[ ];
    const U8            * pOID;
    unsigned             OIDLen;
    CRYPTO_EC_REDUCE_FUNC * pfReduce;
} CRYPTO_EC_CURVE;
```

Structure members

Member	Description
P	Field prime
A	A coefficient
B	B coefficient
G	Generator
Q	Order of curve
OptimizedA	Nonzero if A = -3 (mod P)
aCurveName	Standardized curve name
pOID	Pointer to curve OID octet string
OIDLen	Octet length of the OID octet string
pfReduce	Specialized reduction function

Additional information

Describes the curve $y^2 = x^3 + Ax + B \pmod{P}$

16.2.2 CRYPTO_EC_POINT

Description

Elliptic curve point.

Type definition

```
typedef struct {
    CRYPTO_MPI X;
    CRYPTO_MPI Y;
    CRYPTO_MPI Z;
    CRYPTO_MPI T;
} CRYPTO_EC_POINT;
```

Structure members

Member	Description
X	X coordinate.
Y	Y coordinate.
Z	Nonzero when point is projective.
T	Used by Edwards curves.

Additional information

This type is used for regular and Edwards curves.

16.3 Predefined curves

16.3.1 NIST prime curves

```
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp192r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp192k1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp224r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp224k1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp256r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp256k1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp384r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_secp521r1;
```

16.3.2 Brainpool prime curves

```
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP160r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP160t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP192r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP192t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP224r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP224t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP256r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP256t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP320r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP320t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP384r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP384t1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP512r1;
extern const CRYPTO_EC_CURVE CRYPTO_EC_CURVE_brainpoolP512t1;
```

16.4 Arithmetic

Function	Description
Initialization	
<code>CRYPTO_EC_InitPoint()</code>	Initialize point for use.
<code>CRYPTO_EC_KillPoint()</code>	Zero and then free the memory used to store the values held in <code>Self</code> and then reinitialize <code>pSelf</code> .
<code>CRYPTO_EC_EvictPoint()</code>	Early-kill a point and recover memory.
<code>CRYPTO_EC_MovePoint()</code>	Move <code>pValue</code> to <code>pSelf</code> by destroying <code>pSelf</code> , setting <code>pSelf</code> to <code>pValue</code> , and clear <code>pValue</code> ready for use.
<code>CRYPTO_EC_AssignPoint()</code>	Copy point.
<code>CRYPTO_EC_AssignInf()</code>	Assign point at infinity.
<code>CRYPTO_EC_InitCurve()</code>	Initialize curve for use.
<code>CRYPTO_EC_KillCurve()</code>	Zero and then free the memory used to store the values held in <code>Self</code> and then reinitialize <code>pSelf</code> .
Representation conversions	
<code>CRYPTO_EC_MakeAffine()</code>	Transform projective point to affine coordinates.
<code>CRYPTO_EC_MakeProjective()</code>	Transform affine point to projective coordinates.
Scalar multiplication	
<code>CRYPTO_EC_Mul()</code>	Point scalar multiplication.
<code>CRYPTO_EC_Mul_Basic()</code>	Point multiplication.
<code>CRYPTO_EC_Mul_2b_FW()</code>	Point multiplication, 2-ary, fixed window.
<code>CRYPTO_EC_Mul_3b_FW()</code>	Point multiplication, 3-ary, fixed window.
<code>CRYPTO_EC_Mul_4b_FW()</code>	Point multiplication, 4-ary, fixed window.
<code>CRYPTO_EC_Mul_5b_FW()</code>	Point multiplication, 5-ary, fixed window.
<code>CRYPTO_EC_Mul_6b_FW()</code>	Point multiplication, 6-ary, fixed window.
<code>CRYPTO_EC_Mul_2b_RM()</code>	Point multiplication, 2-ary, fixed window.
<code>CRYPTO_EC_Mul_3b_RM()</code>	Point multiplication, 3-ary, reduced memory.
<code>CRYPTO_EC_Mul_4b_RM()</code>	Point multiplication, 4-ary, reduced memory.
<code>CRYPTO_EC_Mul_5b_RM()</code>	Point multiplication, 5-ary, reduced memory.
<code>CRYPTO_EC_Mul_6b_RM()</code>	Point multiplication, 6-ary, reduced memory.
<code>CRYPTO_EC_Mul_2w_NAF()</code>	Point multiplication, 2b window, nonadjacent form.
<code>CRYPTO_EC_Mul_3w_NAF()</code>	Point multiplication, 3b window, nonadjacent form.
<code>CRYPTO_EC_Mul_4w_NAF()</code>	Point multiplication, 4b window, nonadjacent form.
<code>CRYPTO_EC_Mul_5w_NAF()</code>	Point multiplication, 5b window, nonadjacent form.

Function	Description
<code>CRYPTO_EC_Mul_6w_NAF()</code>	Point multiplication, 6b window, nonadjacent form.
<code>CRYPTO_EC_TwinMul()</code>	Twin point scalar multiplication.
Format conversion	
<code>CRYPTO_EC_WrPointUncompressed()</code>	Encode point, X9.62 uncompressed format.
<code>CRYPTO_EC_WrPointCompressed()</code>	Encode point, X9.62 compressed format.
<code>CRYPTO_EC_WrPointHybrid()</code>	Encode point, X9.62 hybrid format.
Low-level curve arithmetic	
<code>CRYPTO_EC_Add_Affine()</code>	Point addition, affine coordinates.
<code>CRYPTO_EC_Add_Projective()</code>	Point addition, projective coordinates.
<code>CRYPTO_EC_Mul2_Affine()</code>	Point double, affine coordinates.
<code>CRYPTO_EC_Mul2_Projective()</code>	Point double, projective coordinates.
<code>CRYPTO_EC_Mul2Pow_Projective()</code>	Repeated point double, projective coordinates.
<code>CRYPTO_EC_Mul_Affine()</code>	Point scalar multiplication, affine coordinates.
<code>CRYPTO_EC_Mul_Projective()</code>	Point scalar multiplication, projective coordinates.
<code>CRYPTO_EC_Sub_Projective()</code>	Point subtraction, projective coordinates.
Low-level field arithmetic	
<code>CRYPTO_ECC_ModMul()</code>	Field arithmetic, multiply.
<code>CRYPTO_ECC_ModSquare()</code>	Field arithmetic, square.
<code>CRYPTO_ECC_ModDiv()</code>	Field arithmetic, divide.

16.4.1 CRYPTO_ECC_ModDiv()

Description

Field arithmetic, divide.

Prototype

```
int CRYPTO_ECC_ModDiv(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pDivisor,
                           const CRYPTO_EC_CURVE   * pCurve,
                           CRYPTO_MEM_CONTEXT     * pMem);
```

Parameters

Parameter	Description
pSelf	Integer to multiply on entry, product on exit.
pDivisor	Value to divide by.
pCurve	Pointer to elliptic curve group.
pMem	Allocator to use for temporary storage.

Return value

< 0	Error status code.
≥ 0	Success.

Additional information

As CRYPTO_MPI_ModDiv but use any specialized reduction function registered for the curve Curve. If there is no registered reduction function, use generic reduction.

16.4.2 CRYPTO_ECC_ModMul()

Description

Field arithmetic, multiply.

Prototype

```
int CRYPTO_ECC_ModMul(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pMultiplier,
                           const CRYPTO_EC_CURVE   * pCurve,
                           CRYPTO_MEM_CONTEXT     * pMem);
```

Parameters

Parameter	Description
pSelf	Integer to multiply on entry, product on exit.
pMultiplier	Value to multiply by.
pCurve	Pointer to elliptic curve group.
pMem	Allocator to use for temporary storage.

Return value

< 0	Error status code.
≥ 0	Success.

Additional information

As CRYPTO_MPI_ModMul but use any specialized reduction function registered for the curve pCurve. If there is no registered reduction function, use generic reduction.

16.4.3 CRYPTO_ECC_ModSquare()

Description

Field arithmetic, square.

Prototype

```
int CRYPTO_ECC_ModSquare( CRYPTO_MPI * pSelf,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer to square on entry, square on exit.
pCurve	Pointer to elliptic curve group.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

As CRYPTO_MPI_ModSquare but use any specialized reduction function registered for the curve Curve. If there is no registered reduction function, use generic reduction.

16.4.4 CRYPTO_EC_Add_Affine()

Description

Point addition, affine coordinates.

Prototype

```
int CRYPTO_EC_Add_Affine( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_EC_POINT * pValue,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to add to in affine coordinates.
pValue	Point to add to Self in affine coordinates.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.5 CRYPTO_EC_Add_Projective()

Description

Point addition, projective coordinates.

Prototype

```
int CRYPTO_EC_Add_Projective(      CRYPTO_EC_POINT * pSelf,
                                     const CRYPTO_EC_POINT * pValue,
                                     const CRYPTO_EC_CURVE * pCurve,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to add to in projective coordinates.
pValue	Point to add in projective coordinates.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.6 CRYPTO_EC_AssignInf()

Description

Assign point at infinity.

Prototype

```
int CRYPTO_EC_AssignInf(CRYPTO_EC_POINT * pSelf);
```

Parameters

Parameter	Description
pSelf	Point to assign point at infinity.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

The point at infinity is represented by (1, 1, 0).

16.4.7 CRYPTO_EC_AssignPoint()

Description

Copy point.

Prototype

```
int CRYPTO_EC_AssignPoint(      CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_EC_POINT * pValue);
```

Parameters

Parameter	Description
pSelf	Point to assign to.
pValue	Point to copy from.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.8 CRYPTO_EC_EvictPoint()

Description

Early-kill a point and recover memory.

Prototype

```
void CRYPTO_EC_EvictPoint(CRYPTO_EC_POINT * pSelf);
```

Parameters

Parameter	Description
pSelf	Point to destroy.

16.4.9 CRYPTO_EC_InitCurve()

Description

Initialize curve for use.

Prototype

```
void CRYPTO_EC_InitCurve(CRYPTO_EC_CURVE      * pSelf,  
                          CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Curve to initialize.
pMem	Allocator to use for expanding components of curve.

16.4.10 CRYPTO_EC_InitPoint()

Description

Initialize point for use.

Prototype

```
void CRYPTO_EC_InitPoint(CRYPTO_EC_POINT      * pSelf,  
                          CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to initialize.
pMem	Allocator to use for expanding size of pSelf .

16.4.11 CRYPTO_EC_KillCurve()

Description

Zero and then free the memory used to store the values held in Self and then reinitialize pSelf.

Prototype

```
void CRYPTO_EC_KillCurve(CRYPTO_EC_CURVE * pSelf);
```

Parameters

Parameter	Description
pSelf	Curve to destroy.

16.4.12 CRYPTO_EC_KillPoint()

Description

Zero and then free the memory used to store the values held in Self and then reinitialize `pSelf`. You can use this to destroy sensitive key material held in a point.

Prototype

```
void CRYPTO_EC_KillPoint(CRYPTO_EC_POINT * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Point to destroy.

16.4.13 CRYPTO_EC_MakeAffine()

Description

Transform projective point to affine coordinates.

Prototype

```
int CRYPTO_EC_MakeAffine( CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to transform, projective coordinates.
pCurve	Curve the point lies upon.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.14 CRYPTO_EC_MakeProjective()

Description

Transform affine point to projective coordinates.

Prototype

```
int CRYPTO_EC_MakeProjective(CRYPTO_EC_POINT * pSelf);
```

Parameters

Parameter	Description
pSelf	Point to transform.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.15 CRYPTO_EC_MovePoint()

Description

Move `pValue` to `pSelf` by destroying `pSelf`, setting `pSelf` to `pValue`, and clear `pValue` ready for use.

Prototype

```
int CRYPTO_EC_MovePoint(CRYPTO_EC_POINT * pSelf,  
                         CRYPTO_EC_POINT * pValue);
```

Parameters

Parameter	Description
<code>pSelf</code>	Point to assign to.
<code>pValue</code>	Point to move from.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.16 CRYPTO_EC_Mul()

Description

Point scalar multiplication.

Prototype

```
int CRYPTO_EC_Mul(      CRYPTO_EC_POINT    * pSelf,
                        const CRYPTO_MPI     * pK,
                        const CRYPTO_EC_CURVE * pCurve,
                        CRYPTO_MEM_CONTEXT   * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.17 CRYPTO_EC_Mul_2b_FW()

Description

Point multiplication, 2-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_2b_FW( CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_MPI    * pK,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.18 CRYPTO_EC_Mul_3b_FW()

Description

Point multiplication, 3-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_3b_FW( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.19 CRYPTO_EC_Mul_4b_FW()

Description

Point multiplication, 4-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_4b_FW( CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_MPI    * pK,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.20 CRYPTO_EC_Mul_5b_FW()

Description

Point multiplication, 5-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_5b_FW( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.21 CRYPTO_EC_Mul_6b_FW()

Description

Point multiplication, 6-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_6b_FW( CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_MPI    * pK,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.22 CRYPTO_EC_Mul_2b_RM()

Description

Point multiplication, 2-ary, fixed window.

Prototype

```
int CRYPTO_EC_Mul_2b_RM( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.23 CRYPTO_EC_Mul_3b_RM()

Description

Point multiplication, 3-ary, reduced memory.

Prototype

```
int CRYPTO_EC_Mul_3b_RM( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.24 CRYPTO_EC_Mul_4b_RM()

Description

Point multiplication, 4-ary, reduced memory.

Prototype

```
int CRYPTO_EC_Mul_4b_RM( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.25 CRYPTO_EC_Mul_5b_RM()

Description

Point multiplication, 5-ary, reduced memory.

Prototype

```
int CRYPTO_EC_Mul_5b_RM( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0	Error status code.
≥ 0	Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.26 CRYPTO_EC_Mul_6b_RM()

Description

Point multiplication, 6-ary, reduced memory.

Prototype

```
int CRYPTO_EC_Mul_6b_RM( CRYPTO_EC_POINT * pSelf,  
                           const CRYPTO_MPI    * pK,  
                           const CRYPTO_EC_CURVE * pCurve,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.27 CRYPTO_EC_Mul_2w_NAF()

Description

Point multiplication, 2b window, nonadjacent form.

Prototype

```
int CRYPTO_EC_Mul_2w_NAF( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.28 CRYPTO_EC_Mul_3w_NAF()

Description

Point multiplication, 3b window, nonadjacent form.

Prototype

```
int CRYPTO_EC_Mul_3w_NAF( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI   * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.29 CRYPTO_EC_Mul_4w_NAF()

Description

Point multiplication, 4b window, nonadjacent form.

Prototype

```
int CRYPTO_EC_Mul_4w_NAF( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI   * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.30 CRYPTO_EC_Mul_5w_NAF()

Description

Point multiplication, 5b window, nonadjacent form.

Prototype

```
int CRYPTO_EC_Mul_5w_NAF( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.31 CRYPTO_EC_Mul_6w_NAF()

Description

Point multiplication, 6b window, nonadjacent form.

Prototype

```
int CRYPTO_EC_Mul_6w_NAF( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI   * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.32 CRYPTO_EC_Mul_Basic()

Description

Point multiplication.

Prototype

```
int CRYPTO_EC_Mul_Basic( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in affine or projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

Additional information

On return, the multiplied point is returned in the same coordinate system.

16.4.33 CRYPTO_EC_Mul_Affine()

Description

Point scalar multiplication, affine coordinates.

Prototype

```
int CRYPTO_EC_Mul_Affine( CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pK,
                           const CRYPTO_EC_CURVE * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply (in affine coordinates).
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.34 CRYPTO_EC_Mul_Projective()

Description

Point scalar multiplication, projective coordinates.

Prototype

```
int CRYPTO_EC_Mul_Projective( CRYPTO_EC_POINT * pSelf,  
                               const CRYPTO_MPI      * pK,  
                               const CRYPTO_EC_CURVE * pCurve,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to multiply in projective coordinates.
pK	Scalar to multiply pSelf by.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.35 CRYPTO_EC_Mul2_Affine()

Description

Point double, affine coordinates.

Prototype

```
int CRYPTO_EC_Mul2_Affine(      CRYPTO_EC_POINT    * pSelf,
                                const CRYPTO_EC_CURVE   * pCurve,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to double in affine coordinates.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.36 CRYPTO_EC_Mul2_Projective()

Description

Point double, projective coordinates.

Prototype

```
int CRYPTO_EC_Mul2_Projective(      CRYPTO_EC_POINT      * pSelf,  
                                    const CRYPTO_EC_CURVE    * pCurve,  
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to double (in projective coordinates).
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

16.4.37 CRYPTO_EC_Mul2Pow_Projective()

Description

Repeated point double, projective coordinates.

Prototype

```
int CRYPTO_EC_Mul2Pow_Projective( CRYPTO_EC_POINT * pSelf,  
                                    unsigned N,  
                                    const CRYPTO_EC_CURVE * pCurve,  
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to double (in projective coordinates).
N	Count of times to double.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

- ≥ 0 Success.
- < 0 Processing error.

16.4.38 CRYPTO_EC_Sub_Projective()

Description

Point subtraction, projective coordinates.

Prototype

```
int CRYPTO_EC_Sub_Projective( CRYPTO_EC_POINT * pSelf,  
                               CRYPTO_EC_POINT * pValue,  
                               const CRYPTO_EC_CURVE * pCurve,  
                               CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point to subtract from in projective coordinates.
pValue	Point to subtract in projective coordinates.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
≥ 0 Success.

16.4.39 CRYPTO_EC_TwinMul()

Description

Twin point scalar multiplication.

Prototype

```
int CRYPTO_EC_TwinMul(      CRYPTO_EC_POINT * pSelf,
                           const CRYPTO_MPI    * pD0,
                           const CRYPTO_EC_POINT * pS,
                           const CRYPTO_MPI    * pD1,
                           const CRYPTO_EC_POINT * pT,
                           const CRYPTO_EC_CURVE  * pCurve,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Point that receives the sum of products.
pD0	Pointer to scalar to multiply S by.
pS	Pointer to point S.
pD1	Pointer to scalar to multiply T by.
pT	Pointer to point T.
pCurve	Curve that points lie on.
pMem	Allocator to use for temporary storage.

Return value

< 0 Error status code.
 ≥ 0 Success.

Additional information

On return, the sum is returned in projective coordinates.

16.4.40 CRYPTO_EC_WrPointUncompressed()

Description

Encode point, X9.62 uncompressed format.

Prototype

```
void CRYPTO_EC_WrPointUncompressed( CRYPTO_BUFFER * pBuffer,  
                                     const CRYPTO_EC_POINT * pPoint,  
                                     const CRYPTO_EC_CURVE * pCurve);
```

Parameters

Parameter	Description
pBuffer	Pointer to buffer that receives the encoded point.
pPoint	Point to write, affine coordinates.
pCurve	Curve that point lies upon.

16.4.41 CRYPTO_EC_WrPointCompressed()

Description

Encode point, X9.62 compressed format.

Prototype

```
void CRYPTO_EC_WrPointCompressed( CRYPTO_BUFFER * pBuffer,
                                    const CRYPTO_EC_POINT * pPoint,
                                    const CRYPTO_EC_CURVE * pCurve);
```

Parameters

Parameter	Description
pBuffer	Pointer to buffer that receives the encoded point.
pPoint	Point to write, affine coordinates.
pCurve	Curve that point lies upon.

16.4.42 CRYPTO_EC_WrPointHybrid()

Description

Encode point, X9.62 hybrid format.

Prototype

```
void CRYPTO_EC_WrPointHybrid(      CRYPTO_BUFFER     * pBuffer,
                                    const CRYPTO_EC_POINT * pPoint,
                                    const CRYPTO_EC_CURVE * pCurve);
```

Parameters

Parameter	Description
pBuffer	Pointer to buffer that receives the encoded point.
pPoint	Point to write, affine coordinates.
pCurve	Curve that point lies upon.

16.5 Self-test API

The following table lists the elliptic curve self-test API functions.

Function	Description
CRYPTO_EC_NSA_SelfTest()	Run EC self-tests from NSA.
CRYPTO_EC_RFC7027_SelfTest()	Run EC self-tests from RFC 7027.
CRYPTO_EC_SEGGER_SelfTest()	Run EC self-tests from NSA.

16.5.1 CRYPTO_EC_NSA_SelfTest()

Description

Run EC self-tests from NSA.

Prototype

```
void CRYPTO_EC_NSA_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

16.5.2 CRYPTO_EC_RFC7027_SelfTest()

Description

Run EC self-tests from RFC 7027.

Prototype

```
void CRYPTO_EC_RFC7027_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                 CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

16.5.3 CRYPTO_EC_SEGGER_SelfTest()

Description

Run EC self-tests from NSA.

Prototype

```
void CRYPTO_EC_SEGGER_SelfTest(const CRYPTO_SELFTEST_API * pAPI,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pAPI	Pointer to self-test API.
pMem	Pointer to memory allocator for temporary storage.

Chapter 17

Multiprecision integers

17.1 Management

Function	Description
<code>CRYPTO_MPI_Init()</code>	Initialize MPI for use.
<code>CRYPTO_MPI_Kill()</code>	Free the memory used to store the limbs of <code>pSelf</code> and reinitialize <code>pSelf</code> to zero.
<code>CRYPTO_MPI_Evict()</code>	Free the memory used to store the limbs of <code>pSelf</code> and reinitialize <code>pSelf</code> to zero.
<code>CRYPTO_MPI_Reserve()</code>	Preallocate space to hold at least LimbCnt limbs in <code>pSelf</code> .
<code>CRYPTO_MPI_Clear()</code>	Clear MPI.
<code>CRYPTO_MPI_Shrink()</code>	Shrink MPI.

17.1.1 CRYPTO_MPI_Clear()

Description

Clear MPI.

Prototype

```
void CRYPTO_MPI_Clear(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to clear.

Additional information

Note that memory already allocated to the integer `pSelf` is not automatically freed or reduced in size by clearing the value.

17.1.2 CRYPTO_MPI_Evict()

Description

Free the memory used to store the limbs of `pSelf` and reinitialize `pSelf` to zero. Note that the limb data is not guaranteed to be cleared when `pSelf` is destroyed.

Prototype

```
void CRYPTO_MPI_Evict(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to the MPI to evict.

17.1.3 CRYPTO_MPI_Init()

Description

Initialize MPI for use.

Prototype

```
void CRYPTO_MPI_Init(CRYPTO_MPI           * pSelf,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to the MPI to initialize.
pMem	Pointer to the memory allocator that allocates memory as the MPI grows.

17.1.4 CRYPTO_MPI_Kill()

Description

Free the memory used to store the limbs of `pSelf` and reinitialize `pSelf` to zero. Note that the limb data is not guaranteed to be cleared when `pSelf` is destroyed.

Prototype

```
void CRYPTO_MPI_Kill(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to the MPI to free.

17.1.5 CRYPTO_MPI_Reserve()

Description

Preallocate space to hold at least `LimbCnt` limbs in `pSelf`. As multiprecision integers are dynamically extended as required, some higher-level functions will see a performance benefit if they can preallocate space for limbs once rather than extending the integer one limb at a time. For instance, when multiplying two integers, the product width is the sum of the multiplier and multiplicand widths and is, therefore, a candidate for preallocation as the width is computable before calculation begins.

Prototype

```
int CRYPTO_MPI_Reserve(CRYPTO_MPI * pSelf,  
                        unsigned LimbCnt);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to reserve storage for.
<code>LimbCnt</code>	Minimum number of allocated limbs for <code>pSelf</code> .

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.1.6 CRYPTO_MPI_Shrink()

Description

Shrink MPI.

Prototype

```
int CRYPTO_MPI_Shrink(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to shrink.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

This function reduces the memory requirement for an MPI. Some MPIS may get resized up when multiplied and then reduced by the modulus leading to 50% “wasted” space. If your code will deal with only “single-precision” products with reduction, then it may be worthwhile to call `CRYPTO_MPI_Shrink()` to reduce the memory overhead for further processing.

17.2 Assignment

Function	Description
CRYPTO_MPI_Assign()	Assign MPI.
CRYPTO_MPI_AssignInt()	Assign integer.
CRYPTO_MPI_AssignUnsigned()	Assign unsigned.
CRYPTO_MPI_AssignU32()	Assign U32.
CRYPTO_MPI_AssignU64()	Assign U64.
CRYPTO_MPI_Equate()	Equate MPIs.
CRYPTO_MPI_Exchange()	Exchange MPIs.
CRYPTO_MPI_Move()	Move MPI.

17.2.1 CRYPTO_MPI_Assign()

Description

Assign MPI.

Prototype

```
int CRYPTO_MPI_Assign(      CRYPTO_MPI * pSelf,  
                        const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the copy.
pValue	Pointer to value to copy.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Assign `pValue` to `pSelf` by copying.

17.2.2 CRYPTO_MPI_AssignInt()

Description

Assign integer.

Prototype

```
int CRYPTO_MPI_AssignInt(CRYPTO_MPI * pSelf,  
                           int          Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the value.
Value	Value to assign.

Return value

< 0	Processing error.
≥ 0	Success.

17.2.3 CRYPTO_MPI_AssignU32()

Description

Assign U32.

Prototype

```
int CRYPTO_MPI_AssignU32(CRYPTO_MPI * pSelf,  
                           U32           Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the value.
Value	Value to assign.

Return value

≥ 0 Success.
< 0 Processing error.

17.2.4 CRYPTO_MPI_AssignU64()

Description

Assign U64.

Prototype

```
int CRYPTO_MPI_AssignU64(CRYPTO_MPI * pSelf,  
                           U64           Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the value.
Value	Value to assign.

Return value

≥ 0 Success.
< 0 Processing error.

17.2.5 CRYPTO_MPI_AssignUnsigned()

Description

Assign unsigned.

Prototype

```
int CRYPTO_MPI_AssignUnsigned(CRYPTO_MPI * pSelf,  
                               unsigned     Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the value.
Value	Value to assign.

Return value

< 0 Processing error.
≥ 0 Success.

17.2.6 CRYPTO_MPI_Equate()

Description

Equate MPIs.

Prototype

```
void CRYPTO_MPI_Equate(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is assigned to.
pValue	Pointer to the MPI that is assigned.

Additional information

Destroy `pSelf` and make `pSelf` an exact copy of `pValue` and share the limbs. You can use this function to assign key material that is held in read-only memory to an MPI for use in structures such as RSA or DSA public keys.

17.2.7 CRYPTO_MPI_Exchange()

Description

Exchange MPIs.

Prototype

```
void CRYPTO_MPI_Exchange(CRYPTO_MPI * pX,  
                         CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to first MPI.
pY	Pointer to second MPI.

Additional information

The MPIs pointed to by `pX` and `pY` are swapped.

17.2.8 CRYPTO_MPI_Move()

Description

Move MPI.

Prototype

```
int CRYPTO_MPI_Move(CRYPTO_MPI * pSelf,  
                     CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the value.
pValue	Pointer to value to move.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Move `pValue` to `pSelf` by destroying `pSelf`, setting `pSelf` to `pValue`, and clear `pValue` for future for use. `CRYPTO_MPI_Move()` is more efficient than using `MPI_Assign()` followed by a `CRYPTO_MPI_Clear()` to copy values between multiprecision integers.

17.3 Comparisons and predicates

Function	Description
<code>CRYPTO_MPI_IsZero()</code>	Is MPI zero?
<code>CRYPTO_MPI_IsNonzero()</code>	Is MPI nonzero?
<code>CRYPTO_MPI_IsOne()</code>	Is MPI equal to one?
<code>CRYPTO_MPI_IsPositive()</code>	Is MPI positive?
<code>CRYPTO_MPI_IsNegative()</code>	Is MPI negative?
<code>CRYPTO_MPI_IsGreaterZero()</code>	Is MPI strictly positive?
<code>CRYPTO_MPI_IsEven()</code>	Is MPI even (two divides MPI)?
<code>CRYPTO_MPI_IsOdd()</code>	Is MPI odd (two does not divide MPI)?
<code>CRYPTO_MPI_IsEqual()</code>	Is MPI equal to another?
<code>CRYPTO_MPI_IsNotEqual()</code>	Is MPI different from another?
<code>CRYPTO_MPI_IsGreater()</code>	Is MPI greater than another?
<code>CRYPTO_MPI_IsGreaterEqual()</code>	Is MPI greater than or equal to another?
<code>CRYPTO_MPI_IsLess()</code>	Is MPI less than another?
<code>CRYPTO_MPI_IsLessEqual()</code>	Is MPI less than or equal to another?
<code>CRYPTO_MPI_Compare()</code>	Compare MPIS.
<code>CRYPTO_MPI_IsReadOnly()</code>	Query whether MPI is read-only.
<code>CRYPTO_MPI_IsReadWrite()</code>	Query whether MPI is read-write.
<code>CRYPTO_MPI_Sgn()</code>	Calculate signum.
<code>CRYPTO_MPI_Min()</code>	Calculate minimum.
<code>CRYPTO_MPI_Max()</code>	Calculate maximum.

17.3.1 CRYPTO_MPI_Compare()

Description

Compare MPIs.

Prototype

```
int CRYPTO_MPI_Compare(const CRYPTO_MPI * pX,  
                      const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

- < 0 pX is less than pY.
- = 0 pX is equal to pY.
- > 0 pX is greater than pY.

17.3.2 CRYPTO_MPI_IsEqual()

Description

Is MPI equal to another?

Prototype

```
int CRYPTO_MPI_IsEqual(const CRYPTO_MPI * pX,  
                      const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

≠ 0 pX is equal to pY.
= 0 pX is not equal to pY.

17.3.3 CRYPTO_MPI_IsEven()

Description

Is MPI even (two divides MPI)?

Prototype

```
int CRYPTO_MPI_IsEven(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

$\neq 0$ MPI is even.
 $= 0$ MPI is odd.

17.3.4 CRYPTO_MPI_IsGreater()

Description

Is MPI greater than another?

Prototype

```
int CRYPTO_MPI_IsGreater(const CRYPTO_MPI * pX,  
                          const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

$\neq 0$	$pX > pY$
$= 0$	$pX \leq pY$

17.3.5 CRYPTO_MPI_IsGreaterEqual()

Description

Is MPI greater than or equal to another?

Prototype

```
int CRYPTO_MPI_IsGreaterEqual(const CRYPTO_MPI * pX,  
                           const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

$\neq 0$	$pX \geq pY$
$= 0$	$pX < pY$

17.3.6 CRYPTO_MPI_IsGreaterZero()

Description

Is MPI strictly positive?

Prototype

```
int CRYPTO_MPI_IsGreaterZero(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

- $\neq 0$ Argument is greater than zero.
- $= 0$ Argument is less than or equal to zero.

17.3.7 CRYPTO_MPI_IsLess()

Description

Is MPI less than another?

Prototype

```
int CRYPTO_MPI_IsLess(const CRYPTO_MPI * pX,  
                      const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

$\neq 0$	$pX < pY$
$= 0$	$pX \geq pY$

17.3.8 CRYPTO_MPI_IsLessEqual()

Description

Is MPI less than or equal to another?

Prototype

```
int CRYPTO_MPI_IsLessEqual(const CRYPTO_MPI * pX,  
                           const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

$\neq 0$	$pX \leq pY$
$= 0$	$pX > pY$

17.3.9 CRYPTO_MPI_IsNegative()

Description

Is MPI negative?

Prototype

```
int CRYPTO_MPI_IsNegative(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

$\neq 0$ MPI is negative.
 $= 0$ MPI is nonnegative.

Additional information

Zero is always considered positive.

17.3.10 CRYPTO_MPI_IsNonzero()

Description

Is MPI nonzero?

Prototype

```
int CRYPTO_MPI_IsNonzero(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Integer to test.

Return value

$\neq 0$ Argument is nonzero.
 $= 0$ Argument is zero.

17.3.11 CRYPTO_MPI_IsNotEqual()

Description

Is MPI different from another?

Prototype

```
int CRYPTO_MPI_IsNotEqual(const CRYPTO_MPI * pX,  
                           const CRYPTO_MPI * pY);
```

Parameters

Parameter	Description
pX	Pointer to left-hand MPI.
pY	Pointer to right-hand MPI.

Return value

$\neq 0$ pX is equal to pY.
 $= 0$ pX is not equal to pY.

17.3.12 CRYPTO_MPI_IsOdd()

Description

Is MPI odd (two does not divide MPI)?

Prototype

```
int CRYPTO_MPI_IsOdd(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

$\neq 0$ MPI is odd.
 $= 0$ MPI is even.

17.3.13 CRYPTO_MPI_IsOne()

Description

Is MPI equal to one?

Prototype

```
int CRYPTO_MPI_IsOne(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

$\neq 0$ MPI is exactly one.
 $= 0$ MPI is not one.

17.3.14 CRYPTO_MPI_IsPositive()

Description

Is MPI positive?

Prototype

```
int CRYPTO_MPI_IsPositive(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

- = 0 MPI is negative (less than zero).
- ≠ 0 MPI is zero or positive (greater than or equal to zero).

17.3.15 CRYPTO_MPI_IsReadOnly()

Description

Query whether MPI is read-only.

Prototype

```
int CRYPTO_MPI_IsReadOnly(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

- ≠ 0 Object is read-only.
- = 0 Object is read-write.

17.3.16 CRYPTO_MPI_IsReadWrite()

Description

Query whether MPI is read-write.

Prototype

```
int CRYPTO_MPI_IsReadWrite(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

- ≠ 0 Object is read-write.
- = 0 Object is read-only.

17.3.17 CRYPTO_MPI_IsZero()

Description

Is MPI zero?

Prototype

```
int CRYPTO_MPI_IsZero(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Integer to test.

Return value

- ≠ 0 Argument is zero.
- = 0 Argument is nonzero.

17.3.18 CRYPTO_MPI_Max()

Description

Calculate maximum.

Prototype

```
int CRYPTO_MPI_Max(      CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pOther);
```

Parameters

Parameter	Description
pSelf	in Pointer to operand #1. out Maximum of operand #1 and operand #2.
pOther	Pointer to operand #2.

Return value

≥ 0 Success.
< 0 Processing error.

17.3.19 CRYPTO_MPI_Min()

Description

Calculate minimum.

Prototype

```
int CRYPTO_MPI_Min(      CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pOther);
```

Parameters

Parameter	Description
pSelf	in Pointer to operand #1. out Minimum of operand #1 and operand #2.
pOther	Pointer to operand #2.

Return value

≥ 0 Success.
< 0 Processing error.

17.3.20 CRYPTO_MPI_Sgn()

Description

Calculate signum.

Prototype

```
int CRYPTO_MPI_Sgn(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.

Return value

- 1 - MPI is less than zero.
- 0 MPI is zero.
- +1 MPI is greater than zero.

17.4 Addition and subtraction

Function	Description
Addition functions	
<code>CRYPTO_MPI_Add()</code>	Add.
<code>CRYPTO_MPI_AddEx()</code>	Add, three-address form.
<code>CRYPTO_MPI_AddSmall()</code>	Add small value.
<code>CRYPTO_MPI_AddUnsigned()</code>	Add unsigned.
<code>CRYPTO_MPI_Inc()</code>	Add one.
<code>CRYPTO_MPI_Sub()</code>	Subtract.
<code>CRYPTO_MPI_SubUnsigned()</code>	Subtract unsigned.
<code>CRYPTO_MPI_Dec()</code>	Subtract one.
<code>CRYPTO_MPI_RevSub()</code>	Reverse subtract.
<code>CRYPTO_MPI_Neg()</code>	Negate.
<code>CRYPTO_MPI_Abs()</code>	Absolute value.
Modular arithmetic	
<code>CRYPTO_MPI_ModAdd()</code>	Modular add.
<code>CRYPTO_MPI_ModAddEx()</code>	Modular add, three-address form.
<code>CRYPTO_MPI_ModInc()</code>	Modular increment by one.
<code>CRYPTO_MPI_ModSub()</code>	Modular subtract.
<code>CRYPTO_MPI_ModSubEx()</code>	Modular subtract, three-address form.
<code>CRYPTO_MPI_ModDec()</code>	Modular subtract one.
<code>CRYPTO_MPI_ModRevSub()</code>	Modular reverse subtract.
<code>CRYPTO_MPI_ModNeg()</code>	Modular negate.

17.4.1 CRYPTO_MPI_Abs()

Description

Absolute value.

Prototype

```
void CRYPTO_MPI_Abs(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Integer to take magnitude of.

17.4.2 CRYPTO_MPI_Add()

Description

Add.

Prototype

```
int CRYPTO_MPI_Add(      CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to add to.
pValue	Pointer to MPI to be added.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.3 CRYPTO_MPI_AddEx()

Description

Add, three-address form.

Prototype

```
int CRYPTO_MPI_AddEx(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pAugend,  
                           const CRYPTO_MPI * pAddend);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the sum.
pAugend	Pointer to MPI to add to.
pAddend	Pointer to MPI to add.

Return value

< 0 Processing error.
≥ 0 Success.

17.4.4 CRYPTO_MPI_AddSmall()

Description

Add small value.

Prototype

```
int CRYPTO_MPI_AddSmall(CRYPTO_MPI      * pSelf,  
                         CRYPTO_MPI_LIMB  Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to add into.
Value	Small unsigned integer to add.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.5 CRYPTO_MPI_AddUnsigned()

Description

Add unsigned.

Prototype

```
int CRYPTO_MPI_AddUnsigned(CRYPTO_MPI * pSelf,  
                           unsigned     Value);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to add into.
Value	Unsigned integer to add.

Return value

≥ 0 Success.
< 0 Processing error.

17.4.6 CRYPTO_MPI_Dec()

Description

Subtract one.

Prototype

```
int CRYPTO_MPI_Dec(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to decrement.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.7 CRYPTO_MPI_Inc()

Description

Add one.

Prototype

```
int CRYPTO_MPI_Inc(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to increment.

Return value

< 0	Processing error.
≥ 0	Success.

17.4.8 CRYPTO_MPI_ModAdd()

Description

Modular add.

Prototype

```
int CRYPTO_MPI_ModAdd(      CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pValue,
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to add to, $0 \leq \text{Self} < \text{Modulus}$.
pValue	Pointer to MPI to add, $0 \leq \text{Value} < \text{Modulus}$.
pModulus	Pointer to MPI that contains the modulus.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Add two MPIs modulo Modulus. Note the preconditions for the operands: this is an optimized function that requires the operands not exceed the modulus.

17.4.9 CRYPTO_MPI_ModAddEx()

Description

Modular add, three-address form.

Prototype

```
int CRYPTO_MPI_ModAddEx(      CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pX,
                           const CRYPTO_MPI * pY,
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the sum.
pX	Pointer to augend, $0 \leq X < \text{Modulus}$.
pY	Pointer to addend, $0 \leq Y < \text{Modulus}$.
pModulus	Pointer to MPI that contains the modulus.

Return value

≥ 0 Success.
 < 0 Processing error.

Additional information

Add two MPIs modulo Modulus. Note the preconditions for the operands: this is an optimized function that requires the operands not exceed the modulus.

17.4.10 CRYPTO_MPI_ModDec()

Description

Modular subtract one.

Prototype

```
int CRYPTO_MPI_ModDec(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to decrement.
pModulus	Pointer to MPI that contains the modulus.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.11 CRYPTO_MPI_ModInc()

Description

Modular increment by one.

Prototype

```
int CRYPTO_MPI_ModInc(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to increment.
pModulus	Pointer to MPI that contains the modulus.

Return value

≥ 0 Success.
< 0 Processing error.

17.4.12 CRYPTO_MPI_ModNeg()

Description

Modular negate.

Prototype

```
int CRYPTO_MPI_ModNeg(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to negate.
pModulus	Pointer to MPI that contains the modulus.

Return value

≥ 0 Success.
< 0 Processing error.

17.4.13 CRYPTO_MPI_ModSub()

Description

Modular subtract.

Prototype

```
int CRYPTO_MPI_ModSub(      CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pValue,
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to subtract from, $0 \leq \text{Self} < \text{Modulus}$.
pValue	Pointer to MPI to subtract, $0 \leq \text{Value} < \text{Modulus}$.
pModulus	Pointer to MPI that contains the modulus.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Additional information

Subtract Value from Self modulo Modulus. Note the preconditions for the operands: this is an optimized function that requires the operands not exceed the modulus.

17.4.14 CRYPTO_MPI_ModSubEx()

Description

Modular subtract, three-address form.

Prototype

```
int CRYPTO_MPI_ModSubEx(      CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pMinuend,
                           const CRYPTO_MPI * pSubtrahend,
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the difference.
pMinuend	Pointer to MPI to subtract from.
pSubtrahend	Pointer to MPI to subtract.
pModulus	Pointer to MPI that contains the modulus.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.15 CRYPTO_MPI_ModRevSub()

Description

Modular reverse subtract.

Prototype

```
int CRYPTO_MPI_ModRevSub(      CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pValue,
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the value to subtract on entry and assigned the difference on return.
pValue	Pointer to MPI containing the value to subtract from.
pModulus	Pointer to MPI that contains the modulus.

Return value

≥ 0 Success.
 < 0 Processing error.

Additional information

This function subtracts the MPI pointed to by `pValue` from the MPI pointed to by `pSelf` and assigns the difference to the MPI pointed to by `pSelf`.

17.4.16 CRYPTO_MPI_Neg()

Description

Negate.

Prototype

```
void CRYPTO_MPI_Neg(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to negate.

17.4.17 CRYPTO_MPI_RevSub()

Description

Reverse subtract.

Prototype

```
int CRYPTO_MPI_RevSub(      CRYPTO_MPI * pSelf,  
                         const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the value to subtract on entry and assigned the difference on return.
pValue	Pointer to MPI containing the value to subtract from.

Return value

≥ 0 Success.
< 0 Processing error.

Additional information

This function subtracts the MPI pointed to by `pValue` from the MPI pointed to by `pSelf` and assigns the difference to the MPI pointed to by `pSelf`.

17.4.18 CRYPTO_MPI_Sub()

Description

Subtract.

Prototype

```
int CRYPTO_MPI_Sub(      CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to subtract from.
pValue	Pointer to MPI to be subtracted.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.4.19 CRYPTO_MPI_SubUnsigned()

Description

Subtract unsigned.

Prototype

```
int CRYPTO_MPI_SubUnsigned(CRYPTO_MPI * pSelf,  
                           unsigned      Value);
```

Parameters

Parameter	Description
pSelf	Integer to subtract from.
Value	Value to subtract from Self.

Return value

≥ 0 Success.
< 0 Processing error.

17.5 Multiplication

Function	Description
<code>CRYPTO_MPI_Mul()</code>	Multiply.
<code>CRYPTO_MPI_MulEx()</code>	Multiply.
<code>CRYPTO_MPI_Mul2()</code>	Multiply by two.
<code>CRYPTO_MPI_MulUnsigned()</code>	Multiply by unsigned.
<code>CRYPTO_MPI_ShiftLeft()</code>	Multiply by power of two.
<code>CRYPTO_MPI_Square()</code>	Square.
<code>CRYPTO_MPI_SquareEx()</code>	Square, two-operand form.
Modular arithmetic	
<code>CRYPTO_MPI_ModMul()</code>	Modular multiply.
<code>CRYPTO_MPI_ModMulEx()</code>	Modular multiply, three-address form.
<code>CRYPTO_MPI_ModMul2()</code>	Modular multiply by two.
<code>CRYPTO_MPI_ModSquare()</code>	Modular square.
<code>CRYPTO_MPI_ModSquareEx()</code>	Modular square, two-address form.

17.5.1 CRYPTO_MPI_ModSquare()

Description

Modular square.

Prototype

```
int CRYPTO_MPI_ModSquare( CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to square.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.2 CRYPTO_MPI_ModSquareEx()

Description

Modular square, two-address form.

Prototype

```
int CRYPTO_MPI_ModSquareEx( CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pInput,  
                           const CRYPTO_MPI * pModulus,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the square.
pInput	Pointer to MPI to square.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.3 CRYPTO_MPI_ModMul()

Description

Modular multiply.

Prototype

```
int CRYPTO_MPI_ModMul(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pMultiplier,
                           const CRYPTO_MPI        * pModulus,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to multiply.
pMultiplier	Pointer to MPI to multiply by.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.4 CRYPTO_MPI_ModMulEx()

Description

Modular multiply, three-address form.

Prototype

```
int CRYPTO_MPI_ModMulEx( CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pMultiplicand,  
                           const CRYPTO_MPI * pMultiplier,  
                           const CRYPTO_MPI * pModulus,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the product.
pMultiplicand	Pointer to MPI to multiply.
pMultiplier	Pointer to MPI to multiply by.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.5 CRYPTO_MPI_ModMul2()

Description

Modular multiply by two.

Prototype

```
int CRYPTO_MPI_ModMul2(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Value to double. A precondition is that $0 \leq \text{pSelf} < \text{Modulus}$, i.e. the input is already reduced modulo the modulus.
pModulus	Pointer to MPI that contains the modulus.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.6 CRYPTO_MPI_Mul()

Description

Multiply.

Prototype

```
int CRYPTO_MPI_Mul(      CRYPTO_MPI          * pSelf,
                        const CRYPTO_MPI        * pMultiplier,
                        CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to multiply.
pMultiplier	Pointer to MPI to multiply by.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.7 CRYPTO_MPI_Mul2()

Description

Multiply by two.

Prototype

```
int CRYPTO_MPI_Mul2(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to be multiplied.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.8 CRYPTO_MPI_MulEx()

Description

Multiply.

Prototype

```
int CRYPTO_MPI_MulEx(      CRYPTO_MPI      * pSelf,
                           const CRYPTO_MPI   * pMultiplicand,
                           const CRYPTO_MPI   * pMultiplier,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the product.
pMultiplicand	Pointer to MPI to multiply.
pMultiplier	Pointer to MPI to multiply by.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.9 CRYPTO_MPI_MulUnsigned()

Description

Multiply by unsigned.

Prototype

```
int CRYPTO_MPI_MulUnsigned(CRYPTO_MPI           * pSelf,  
                           unsigned            Multiplier,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to multiply.
Multiplier	Unsigned multiplier.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.10 CRYPTO_MPI_ShiftLeft()

Description

Multiply by power of two.

Prototype

```
int CRYPTO_MPI_ShiftLeft(CRYPTO_MPI * pSelf,  
                           unsigned      BitCnt);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to shift.
BitCnt	Number of bit positions to shift.

Return value

< 0 Processing error.
≥ 0 Success.

17.5.11 CRYPTO_MPI_Square()

Description

Square.

Prototype

```
int CRYPTO_MPI_Square(CRYPTO_MPI           * pSelf,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to square.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0	Processing error.
≥ 0	Success.

17.5.12 CRYPTO_MPI_SquareEx()

Description

Square, two-operand form.

Prototype

```
int CRYPTO_MPI_SquareEx(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI           * pInput,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the square.
pInput	Pointer to MPI to square.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.6 Division

Function	Description
<code>CRYPTO_MPI_Div()</code>	Divide.
<code>CRYPTO_MPI_Div2()</code>	Divide by two.
<code>CRYPTO_MPI_DivUnsigned()</code>	Divide by unsigned.
<code>CRYPTO_MPI_ShiftRight()</code>	Divide by power of two.
<code>CRYPTO_MPI_DivMod()</code>	Divide and return remainder.
<code>CRYPTO_MPI_RevDiv()</code>	Reverse divide.
<code>CRYPTO_MPI_CeilDiv()</code>	Ceiling divide.
<code>CRYPTO_MPI_Mod()</code>	Remainder after division.
<code>CRYPTO_MPI_ModEx()</code>	Remainder after division, three-address form.
<code>CRYPTO_MPI_ModUnsigned()</code>	Remainder after division by unsigned.
<code>CRYPTO_MPI_Sqrt()</code>	Square root.
Modular arithmetic	
<code>CRYPTO_MPI_ModDiv()</code>	Modular divide.
<code>CRYPTO_MPI_ModDiv2()</code>	Modular divide by two.
<code>CRYPTO_MPI_ModInv()</code>	Modular inverse.
<code>CRYPTO_MPI_ModInvEx()</code>	Modular inverse, two-address form.
<code>CRYPTO_MPI_ModSqrt()</code>	Modular square root.

17.6.1 CRYPTO_MPI_CeilDiv()

Description

Ceiling divide.

Prototype

```
int CRYPTO_MPI_CeilDiv(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pDivisor,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the dividend and receiving the quotient.
pDivisor	Pointer to MPI containing the divisor.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Computes Self = Ceil(Self / Divisor).

17.6.2 CRYPTO_MPI_Div()

Description

Divide.

Prototype

```
int CRYPTO_MPI_Div(      CRYPTO_MPI      * pSelf,
                        const CRYPTO_MPI      * pDivisor,
                        CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to divide.
pDivisor	Pointer to MPI to divide by.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Divide `pSelf` by `pDivisor`, i.e. `Self /= Divisor`.

17.6.3 CRYPTO_MPI_Div2()

Description

Divide by two.

Prototype

```
int CRYPTO_MPI_Div2(CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to halve.

Return value

Remainder after division, 0 or 1.

Additional information

Shifts the magnitude of `pSelf` right by one bit and return the bit carried out.

This function can never fail with an error code.

17.6.4 CRYPTO_MPI_DivMod()

Description

Divide and return remainder.

Prototype

```
int CRYPTO_MPI_DivMod(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pDivisor,
                           CRYPTO_MPI            * pRemainder,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the dividend and receiving the quotient.
pDivisor	Pointer to MPI containing the divisor.
pRemainder	Pointer to MPI that receives the remainder.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.6.5 CRYPTO_MPI_DivUnsigned()

Description

Divide by unsigned.

Prototype

```
int CRYPTO_MPI_DivUnsigned(CRYPTO_MPI           * pSelf,  
                           unsigned             Divisor,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing dividend and receiving the quotient.
Divisor	Unsigned divisor.
pMem	Pointer to allocator that provides temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.6.6 CRYPTO_MPI_Mod()

Description

Remainder after division.

Prototype

```
int CRYPTO_MPI_Mod(      CRYPTO_MPI      * pSelf,
                      const CRYPTO_MPI      * pDivisor,
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the dividend and receiving the remainder.
pDivisor	Pointer to MPI containing the divisor.
pMem	Pointer to allocator that provides temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.6.7 CRYPTO_MPI_ModDiv()

Description

Modular divide.

Prototype

```
int CRYPTO_MPI_ModDiv(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pDivisor,
                           const CRYPTO_MPI        * pModulus,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to divide.
pDivisor	Pointer to MPI to divide by.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Divide Self by Divisor (mod Modulus). This is equivalent to multiplying Self by the modular inverse of Divisor (mod Modulus).

17.6.8 CRYPTO_MPI_ModDiv2()

Description

Modular divide by two.

Prototype

```
int CRYPTO_MPI_ModDiv2(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pModulus);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to halve. A precondition is that $0 \leq \text{Value} < \text{Modulus}$, i.e. the input is already reduced modulo Modulus.
pModulus	Pointer to MPI that contains the modulus.

Return value

- ≥ 0 Success.
- < 0 Processing error.

17.6.9 CRYPTO_MPI_ModEx()

Description

Remainder after division, three-address form.

Prototype

```
int CRYPTO_MPI_ModEx( CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pDividend,  
                      const CRYPTO_MPI * pDivisor,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the remainder.
pDividend	Pointer to MPI to divide.
pDivisor	Pointer to MPI to divide by.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.6.10 CRYPTO_MPI_ModInv()

Description

Modular inverse.

Prototype

```
int CRYPTO_MPI_ModInv(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pModulus,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to invert.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

≥ 0	Success.
= CRYPTO_ERROR_NO_INVERSE	No modular inverse exists.
< 0	Processing error (includes no modular inverse).

Additional information

This implementation conforms to [FIPS186] and the standard states that a FIPS-conforming implementation of the modular inverse must use the algorithm in section C.1.

17.6.11 CRYPTO_MPI_ModInvEx()

Description

Modular inverse, two-address form.

Prototype

```
int CRYPTO_MPI_ModInvEx( CRYPTO_MPI * pSelf,
                           const CRYPTO_MPI * pValue,
                           const CRYPTO_MPI * pModulus,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the modular inverse.
pValue	Pointer to MPI to be inverted.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to allocator that provides temporary storage.

Return value

≥ 0	Success.
= CRYPTO_ERROR_NO_INVERSE	No modular inverse exists.
< 0	Processing error (includes no modular inverse).

Additional information

This implementation conforms to [FIPS186] and the standard states that a FIPS-conforming implementation of the modular inverse must use the algorithm in section C.1.

17.6.12 CRYPTO_MPI_ModSqrt()

Description

Modular square root.

Prototype

```
int CRYPTO_MPI_ModSqrt( CRYPTO_MPI * pSelf,
                         const CRYPTO_MPI * pModulus,
                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.
pModulus	Pointer to MPI that contains the modulus.
pMem	Pointer to temporary storage allocator context.

Return value

≥ 0	Success.
= CRYPTO_ERROR_NO_SQUARE_ROOT	Square root does not exist.
< 0	Processing error (includes no square root).

Additional information

The complementary square root is easily calculated by negating the returned square root using `CRYPTO_MPI_ModNeg()`.

17.6.13 CRYPTO_MPI_ModUnsigned()

Description

Remainder after division by unsigned.

Prototype

```
int CRYPTO_MPI_ModUnsigned(CRYPTO_MPI           * pSelf,  
                           unsigned             Divisor,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI containing the dividend and receiving the remainder.
Divisor	Unsigned divider.
pMem	Pointer to allocator that provides temporary storage.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.6.14 CRYPTO_MPI_RevDiv()

Description

Reverse divide.

Prototype

```
int CRYPTO_MPI_RevDiv(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pDividend,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	in Pointer to MPI that contains the divisor. out Pointer to MPI that contains the quotient.
pDividend	Pointer to MPI to divide by.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Sets Self to Dividend divided by Self.

17.6.15 CRYPTO_MPI_ShiftRight()

Description

Divide by power of two.

Prototype

```
void CRYPTO_MPI_ShiftRight(CRYPTO_MPI * pSelf,  
                           unsigned      BitCnt);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to shift.
BitCnt	Number of bit positions to shift by.

17.6.16 CRYPTO_MPI_Sqrt()

Description

Square root.

Prototype

```
int CRYPTO_MPI_Sqrt(CRYPTO_MPI           * pSelf,  
                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to root.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.7 Exponentiation

Function	Description
<code>CRYPTO_MPI_2Exp()</code>	Assign power of two.
<code>CRYPTO_MPI_2ExpMinusOne()</code>	Create bitmask.
<code>CRYPTO_MPI_Exp()</code>	Exponentiate.
Modular arithmetic, generic	
<code>CRYPTO_MPI_ModExp()</code>	Modular exponentiate.
<code>CRYPTO_MPI_ModExp2Pow()</code>	Modular exponentiate by power of two.
Basic algorithm	
<code>CRYPTO_MPI_ModExp_Basic_Fast()</code>	Modular exponentiate, fast.
<code>CRYPTO_MPI_ModExp_Basic_Ladder()</code>	Modular exponentiate, Montgomery ladder.
<code>CRYPTO_MPI_ModExp_Basic_2b_FW()</code>	Modular exponentiate, 2-bit window.
<code>CRYPTO_MPI_ModExp_Basic_3b_FW()</code>	Modular exponentiate, 3-bit window.
<code>CRYPTO_MPI_ModExp_Basic_4b_FW()</code>	Modular exponentiate, 4-bit window.
<code>CRYPTO_MPI_ModExp_Basic_5b_FW()</code>	Modular exponentiate, 5-bit window.
<code>CRYPTO_MPI_ModExp_Basic_6b_FW()</code>	Modular exponentiate, 6-bit window.
<code>CRYPTO_MPI_ModExp_Basic_2b_RM()</code>	Modular exponentiate, 2-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Basic_3b_RM()</code>	Modular exponentiate, 3-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Basic_4b_RM()</code>	Modular exponentiate, 4-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Basic_5b_RM()</code>	Modular exponentiate, 5-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Basic_6b_RM()</code>	Modular exponentiate, 6-bit window, reduced memory.
Windowing, Montgomery reduction	
<code>CRYPTO_MPI_ModExp_Montgomery_Fast()</code>	Modular exponentiate, fast, Montgomery reduction.
<code>CRYPTO_MPI_ModExp_Montgomery_Ladder()</code>	Modular exponentiate, Montgomery ladder, Montgomery reduction.
<code>CRYPTO_MPI_ModExp_Montgomery_2b_FW()</code>	Modular exponentiate, Montgomery reduce, 2-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_3b_FW()</code>	Modular exponentiate, Montgomery reduce, 3-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_4b_FW()</code>	Modular exponentiate, Montgomery reduce, 4-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_5b_FW()</code>	Modular exponentiate, Montgomery reduce, 5-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_6b_FW()</code>	Modular exponentiate, Montgomery reduce, 6-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_2b_RM()</code>	Modular exponentiate, Montgomery reduce, 2-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Montgomery_3b_RM()</code>	Modular exponentiate, Montgomery reduce, 3-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Montgomery_4b_RM()</code>	Modular exponentiate, Montgomery reduce, 4-bit window, reduced memory.

Function	Description
<code>CRYPTO_MPI_ModExp_Montgomery_5b_RM()</code>	Modular exponentiate, Montgomery reduce, 5-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Montgomery_6b_RM()</code>	Modular exponentiate, Montgomery reduce, 6-bit window, reduced memory.
Windowing, Barrett reduction	
<code>CRYPTO_MPI_ModExp_Barrett_Fast()</code>	Modular exponentiate, fast, Barrett reduce.
<code>CRYPTO_MPI_ModExp_Barrett_Ladder()</code>	Modular exponentiate, Montgomery ladder, Barrett reduce.
<code>CRYPTO_MPI_ModExp_Barrett_2b_FW()</code>	Modular exponentiate, Barrett reduce, 2-bit window.
<code>CRYPTO_MPI_ModExp_Barrett_3b_FW()</code>	Modular exponentiate, Barrett reduce, 3-bit window.
<code>CRYPTO_MPI_ModExp_Barrett_4b_FW()</code>	Modular exponentiate, Barrett reduce, 4-bit window.
<code>CRYPTO_MPI_ModExp_Barrett_5b_FW()</code>	Modular exponentiate, Barrett reduce, 5-bit window.
<code>CRYPTO_MPI_ModExp_Barrett_6b_FW()</code>	Modular exponentiate, Barrett reduce, 6-bit window.
<code>CRYPTO_MPI_ModExp_Barrett_2b_RM()</code>	Modular exponentiate, Barrett reduce, 2-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Barrett_3b_RM()</code>	Modular exponentiate, Barrett reduce, 3-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Barrett_4b_RM()</code>	Modular exponentiate, Barrett reduce, 4-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Barrett_5b_RM()</code>	Modular exponentiate, Barrett reduce, 5-bit window, reduced memory.
<code>CRYPTO_MPI_ModExp_Barrett_6b_RM()</code>	Modular exponentiate, Barrett reduce, 6-bit window, reduced memory.

17.7.1 CRYPTO_MPI_2Exp()

Description

Assign power of two.

Prototype

```
int CRYPTO_MPI_2Exp(CRYPTO_MPI * pSelf,  
                      unsigned      Exponent);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the power of two.
Exponent	Power of two.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Assign 2^{Exponent} to Self.

17.7.2 CRYPTO_MPI_2ExpMinusOne()

Description

Create bitmask.

Prototype

```
int CRYPTO_MPI_2ExpMinusOne(CRYPTO_MPI * pSelf,  
                           unsigned      BitCnt);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the bitmask.
BitCnt	Power-of-two exponent.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

This sets `pSelf` to $2^{\text{BitCnt}} - 1$, creating a mask with all bits between 0 and `BitCnt`-1 inclusive set to one.

17.7.3 CRYPTO_MPI_Exp()

Description

Exponentiate.

Prototype

```
int CRYPTO_MPI_Exp( CRYPTO_MPI * pSelf,  
                     const CRYPTO_MPI * pExponent,  
                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MPI that is the base.
<code>pExponent</code>	Pointer to MPI that is the exponent.
<code>pMem</code>	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

This raises the base, pointed to by `pSelf`, to the power of `pExponent`.

17.7.4 CRYPTO_MPI_ModExp()

Description

Modular exponentiate.

Prototype

```
int CRYPTO_MPI_ModExp(      CRYPTO_MPI          * pSelf,
                           const CRYPTO_MPI        * pExponent,
                           const CRYPTO_MPI        * pModulus,
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is the base.
pExponent	Pointer to MPI that is the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

This raises the base, pointed to by `pSelf`, to the power of `pExponent` and reduces the result modulo `pModulus`.

17.7.5 CRYPTO_MPI_ModExp2Pow()

Description

Modular exponentiate by power of two.

Prototype

```
int CRYPTO_MPI_ModExp2Pow( CRYPTO_MPI * pSelf,
                            unsigned PowerOfTwo,
                            const CRYPTO_MPI * pModulus,
                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is the base.
PowerOfTwo	Power-of-two exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
 ≥ 0 Success.

Additional information

This raises the base, pointed to by `pSelf`, to the power of $2^{\text{PowerOfTwo}}$, and reduces the result modulo `pModulus`.

17.7.6 CRYPTO_MPI_ModExp_Basic_Fast()

Description

Modular exponentiate, fast.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_Fast( CRYPTO_MPI * pSelf,
                                    const CRYPTO_MPI * pExponent,
                                    const CRYPTO_MPI * pModulus,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error.
≥ 0	Success.

Additional information

This implementation uses a Montgomery ladder to defend against side channel attacks.

17.7.7 CRYPTO_MPI_ModExp_Basic_Ladder()

Description

Modular exponentiate, Montgomery ladder.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_Ladder( CRYPTO_MPI * pSelf,
                                      const CRYPTO_MPI * pExponent,
                                      const CRYPTO_MPI * pModulus,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

This implementation uses a Montgomery ladder to defend against side channel attacks.

17.7.8 CRYPTO_MPI_ModExp_Basic_2b_FW()

Description

Modular exponentiate, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_2b_FW( CRYPTO_MPI * pSelf,
                                     const CRYPTO_MPI * pExponent,
                                     const CRYPTO_MPI * pModulus,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.9 CRYPTO_MPI_ModExp_Basic_3b_FW()

Description

Modular exponentiate, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_3b_FW( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.10 CRYPTO_MPI_ModExp_Basic_4b_FW()

Description

Modular exponentiate, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_4b_FW( CRYPTO_MPI * pSelf,
                                     const CRYPTO_MPI * pExponent,
                                     const CRYPTO_MPI * pModulus,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.11 CRYPTO_MPI_ModExp_Basic_5b_FW()

Description

Modular exponentiate, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_5b_FW( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.12 CRYPTO_MPI_ModExp_Basic_6b_FW()

Description

Modular exponentiate, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_6b_FW( CRYPTO_MPI * pSelf,
                                     const CRYPTO_MPI * pExponent,
                                     const CRYPTO_MPI * pModulus,
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.13 CRYPTO_MPI_ModExp_Basic_2b_RM()

Description

Modular exponentiate, 2-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_2b_RM( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.14 CRYPTO_MPI_ModExp_Basic_3b_RM()

Description

Modular exponentiate, 3-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_3b_RM( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.15 CRYPTO_MPI_ModExp_Basic_4b_RM()

Description

Modular exponentiate, 4-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_4b_RM( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.16 CRYPTO_MPI_ModExp_Basic_5b_RM()

Description

Modular exponentiate, 5-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_5b_RM( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.17 CRYPTO_MPI_ModExp_Basic_6b_RM()

Description

Modular exponentiate, 6-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Basic_6b_RM( CRYPTO_MPI * pSelf,  
                                     const CRYPTO_MPI * pExponent,  
                                     const CRYPTO_MPI * pModulus,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.18 CRYPTO_MPI_ModExp_Barrett_Fast()

Description

Modular exponentiate, fast, Barrett reduce.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_Fast( CRYPTO_MPI * pSelf,
                                      const CRYPTO_MPI * pExponent,
                                      const CRYPTO_MPI * pModulus,
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

17.7.19 CRYPTO_MPI_ModExp_Barrett_Ladder()

Description

Modular exponentiate, Montgomery ladder, Barrett reduce.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_Ladder( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

17.7.20 CRYPTO_MPI_ModExp_Barrett_2b_FW()

Description

Modular exponentiate, Barrett reduce, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_2b_FW( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.21 CRYPTO_MPI_ModExp_Barrett_3b_FW()

Description

Modular exponentiate, Barrett reduce, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_3b_FW( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.22 CRYPTO_MPI_ModExp_Barrett_4b_FW()

Description

Modular exponentiate, Barrett reduce, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_4b_FW( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.23 CRYPTO_MPI_ModExp_Barrett_5b_FW()

Description

Modular exponentiate, Barrett reduce, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_5b_FW( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.24 CRYPTO_MPI_ModExp_Barrett_6b_FW()

Description

Modular exponentiate, Barrett reduce, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_6b_FW( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.25 CRYPTO_MPI_ModExp_Barrett_2b_RM()

Description

Modular exponentiate, Barrett reduce, 2-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_2b_RM( CRYPTO_MPI * pSelf,
                                       const CRYPTO_MPI * pExponent,
                                       const CRYPTO_MPI * pModulus,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.26 CRYPTO_MPI_ModExp_Barrett_3b_RM()

Description

Modular exponentiate, Barrett reduce, 3-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_3b_RM( CRYPTO_MPI * pSelf,
                                       const CRYPTO_MPI * pExponent,
                                       const CRYPTO_MPI * pModulus,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.27 CRYPTO_MPI_ModExp_Barrett_4b_RM()

Description

Modular exponentiate, Barrett reduce, 4-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_4b_RM( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.28 CRYPTO_MPI_ModExp_Barrett_5b_RM()

Description

Modular exponentiate, Barrett reduce, 5-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_5b_RM( CRYPTO_MPI * pSelf,
                                       const CRYPTO_MPI * pExponent,
                                       const CRYPTO_MPI * pModulus,
                                       CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.29 CRYPTO_MPI_ModExp_Barrett_6b_RM()

Description

Modular exponentiate, Barrett reduce, 6-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Barrett_6b_RM( CRYPTO_MPI * pSelf,  
                                      const CRYPTO_MPI * pExponent,  
                                      const CRYPTO_MPI * pModulus,  
                                      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.30 CRYPTO_MPI_ModExp_Montgomery_Fast()

Description

Modular exponentiate, fast, Montgomery reduction.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_Fast( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error.
≥ 0 Success.

17.7.31 CRYPTO_MPI_ModExp_Montgomery_Ladder()

Description

Modular exponentiate, Montgomery ladder, Montgomery reduction.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_Ladder( CRYPTO_MPI * pSelf,
                                         const CRYPTO_MPI * pExponent,
                                         const CRYPTO_MPI * pModulus,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error.
≥ 0	Success.

17.7.32 CRYPTO_MPI_ModExp_Montgomery_2b_FW()

Description

Modular exponentiate, Montgomery reduce, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_FW( CRYPTO_MPI * pSelf,
                                         const CRYPTO_MPI * pExponent,
                                         const CRYPTO_MPI * pModulus,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.33 CRYPTO_MPI_ModExp_Montgomery_3b_FW()

Description

Modular exponentiate, Montgomery reduce, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_FW( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.34 CRYPTO_MPI_ModExp_Montgomery_4b_FW()

Description

Modular exponentiate, Montgomery reduce, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_FW( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.35 CRYPTO_MPI_ModExp_Montgomery_5b_FW()

Description

Modular exponentiate, Montgomery reduce, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_FW( CRYPTO_MPI * pSelf,
                                         const CRYPTO_MPI * pExponent,
                                         const CRYPTO_MPI * pModulus,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.36 CRYPTO_MPI_ModExp_Montgomery_6b_FW()

Description

Modular exponentiate, Montgomery reduce, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_FW( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.37 CRYPTO_MPI_ModExp_Montgomery_2b_RM()

Description

Modular exponentiate, Montgomery reduce, 2-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_RM( CRYPTO_MPI * pSelf,
                                         const CRYPTO_MPI * pExponent,
                                         const CRYPTO_MPI * pModulus,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.38 CRYPTO_MPI_ModExp_Montgomery_3b_RM()

Description

Modular exponentiate, Montgomery reduce, 3-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_RM( CRYPTO_MPI * pSelf,
                                         const CRYPTO_MPI * pExponent,
                                         const CRYPTO_MPI * pModulus,
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

17.7.39 CRYPTO_MPI_ModExp_Montgomery_4b_RM()

Description

Modular exponentiate, Montgomery reduce, 4-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_RM( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.40 CRYPTO_MPI_ModExp_Montgomery_5b_RM()

Description

Modular exponentiate, Montgomery reduce, 5-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_RM( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.7.41 CRYPTO_MPI_ModExp_Montgomery_6b_RM()

Description

Modular exponentiate, Montgomery reduce, 6-bit window, reduced memory.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_RM( CRYPTO_MPI * pSelf,  
                                         const CRYPTO_MPI * pExponent,  
                                         const CRYPTO_MPI * pModulus,  
                                         CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

17.8 Bit and byte-level access

Function	Description
<code>CRYPTO_MPI_RdBit()</code>	Query the value of a single bit in an MPI.
<code>CRYPTO_MPI_RdBits()</code>	Read bitslice from MPI.
<code>CRYPTO_MPI_RdByte()</code>	Read byte from MPI.
<code>CRYPTO_MPI_ExtractBits()</code>	Copy bitslice of MPI.
<code>CRYPTO_MPI_SetBit()</code>	Set bit BitNumber in pSelf to one.
<code>CRYPTO_MPI_ClrBit()</code>	Set bit BitNumber in pSelf to zero.
<code>CRYPTO_MPI_WrBit()</code>	Set bit BitNumber in pSelf to Value.
<code>CRYPTO_MPI_TrimBits()</code>	Clear MPI high order bits.
<code>CRYPTO_MPI_TrimLimbs()</code>	Clear MPI high order bits.
<code>CRYPTO_MPI_BitCount()</code>	Query the number of bits required to represent an MPI.
<code>CRYPTO_MPI_ByteCount()</code>	Inquire octet length of MPI.
<code>CRYPTO_MPI_ByteCount_ASN1()</code>	Inquire octet length for ASN.1 encoding.
<code>CRYPTO_MPI_LSB()</code>	Return the bit number of the least significant nonzero bit in the magnitude of Self.
<code>CRYPTO_MPI_MSB()</code>	Return the bit number of the most significant nonzero bit in the magnitude of Self.
<code>CRYPTO_MPI_Unsigned()</code>	Return the least significant limb as an unsigned value, without respecting the sign of the argument.
<code>CRYPTO_MPI_LimbsRequired()</code>	Compute limbs required to hold value.

17.8.1 CRYPTO_MPI_BitCount()

Description

Query the number of bits required to represent an MPI. For instance, the value 6 requires 3 bits, 1023 requires 9 bits, and 1024 requires 10 bits.

Prototype

```
unsigned CRYPTO_MPI_BitCount(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

Number of bits used to represent the MPI.

Additional information

Although this is related to the `MPI_MSB()` function, the two are not trivially equivalent. For nonzero Self, the two are related by `CRYPTTO_MSB(x) = CRYPTTO_BitCount(x)-1`, but they differ when x is zero. In this case, both return zero.

17.8.2 CRYPTO_MPI_ByteCount()

Description

Inquire octet length of MPI.

Prototype

```
unsigned CRYPTO_MPI_ByteCount(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

Octet length of the MPI.

Additional information

The octet length of an MPI is the number of bytes required to represent the integer in an octet string. This function does not consider the sign of the most significant byte in the octet string, for that use CRYPTO_MPI_ByteCountN().

17.8.3 CRYPTO_MPI_ByteCount ASN1()

Description

Inquire octet length for ASN.1 encoding.

Prototype

```
unsigned CRYPTO_MPI_ByteCount ASN1(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

Octet length of the MPI for ASN.1 encoding.

Additional information

The octet length of an MPI is the number of bytes required to represent the integer in an octet string where the most significant bit of the octet string represents the sign bit.

If the most significant bit of the MPI corresponds to bit 7 of the leading octet, an ASN.1 reader will interpret this as a negative value, and hence we increase the length of the MPI representation by one octet in order that the leading bit is a zero.

17.8.4 CRYPTO_MPI_ExtractBits()

Description

Copy bitslice of MPI.

Prototype

```
int CRYPTO_MPI_ExtractBits(      CRYPTO_MPI * pSelf,
                                const CRYPTO_MPI * pSource,
                                unsigned          LowBit,
                                unsigned          Width);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MPI that receives the bitslice.
<code>pSource</code>	Pointer to MPI that is the source MPI.
<code>LowBit</code>	Low-order bit index to extract.
<code>Width</code>	<code>Width</code> of bitslice, i.e. number of bits.

Return value

Bitslice of MPI from low-order bit index `LowBit` to high order bit index `LowBit+Width-1`.

- < 0 Processing error.
- ≥ 0 Success.

17.8.5 CRYPTO_MPI_WrBit()

Description

Set bit BitNumber in `pSelf` to `Value`. It is acceptable to set bytes beyond the end of the integer, in which case the integer is extended.

Prototype

```
int CRYPTO_MPI_WrBit(CRYPTO_MPI * pSelf,  
                      unsigned     BitIndex,  
                      int          Value);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to MPI.
<code>BitIndex</code>	Bit number to store into, 0 is lsb.
<code>Value</code>	<code>Value</code> of bit to store (0 stores zero, nonzero stores 1).

Return value

< 0	Processing error.
≥ 0	Success.

17.8.6 CRYPTO_MPI_ClrBit()

Description

Set bit BitNumber in `pSelf` to zero. It is acceptable to set bytes beyond the end of the integer, in which case the integer is extended.

Prototype

```
void CRYPTO_MPI_ClrBit(CRYPTO_MPI * pSelf,  
                        unsigned     BitIndex);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to store into.
<code>BitIndex</code>	Bit number to set to zero, 0 is lsb.

17.8.7 CRYPTO_MPI_RdBit()

Description

Query the value of a single bit in an MPI. It is acceptable to inquire bits beyond the end of the MPI, in which case the bit is returned as zero.

Prototype

```
unsigned CRYPTO_MPI_RdBit(const CRYPTO_MPI * pSelf,  
                           unsigned     BitIndex);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.
BitIndex	Bit number of MPI to query.

Return value

Value of bit within MPI.

17.8.8 CRYPTO_MPI_RdBits()

Description

Read bitslice from MPI.

Prototype

```
U32 CRYPTO_MPI_RdBits(const CRYPTO_MPI * pSelf,  
                      unsigned BitIndex,  
                      unsigned Width);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is sliced.
BitIndex	Index of low order bit.
Width	Number of bits to extract, must be less than or equal to 32.

Return value

Bitslice of MPI from low-order bit index `BitIndex` to high order bit index `BitIndex+Width-1`.

17.8.9 CRYPTO_MPI_RdByte()

Description

Read byte from MPI.

Prototype

```
U8 CRYPTO_MPI_RdByte(const CRYPTO_MPI * pSelf,  
                      unsigned ByteIndex);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.
ByteIndex	Byte index to read.

Return value

The byte at the given index.

Additional information

Byte 0 is bits 0 through 7, byte 1 is 8 through 15, and so on. It is acceptable to read bytes beyond the end of the MPI in which case the additional bits are read a zero.

17.8.10 CRYPTO_MPI_SetBit()

Description

Set bit BitNumber in `pSelf` to one. It is acceptable to set bytes beyond the end of the integer, in which case the integer is extended.

Prototype

```
int CRYPTO_MPI_SetBit(CRYPTO_MPI * pSelf,  
                      unsigned     BitIndex);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to store into.
<code>BitIndex</code>	Bit number to set to one, 0 is lsb.

Return value

< 0 Processing error.
≥ 0 Success.

17.8.11 CRYPTO_MPI_TrimBits()

Description

Clear MPI high order bits.

Prototype

```
void CRYPTO_MPI_TrimBits(CRYPTO_MPI * pSelf,  
                           unsigned     BitWidth);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is trimmed.
BitWidth	Number of low-order bits to retain in the MPI.

Additional information

Clear all bits of `pSelf` with indexes greater than or equal to `BitWidth`, i.e. set `pSelf = pSelf mod 2^BitWidth`.

17.8.12 CRYPTO_MPI_TrimLimbs()

Description

Clear MPI high order bits.

Prototype

```
void CRYPTO_MPI_TrimLimbs(CRYPTO_MPI * pSelf,  
                           unsigned LimbCnt);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that is trimmed.
LimbCnt	Number of low-order limbs to retain in the MPI.

Additional information

Clear all bits of `pSelf` with indexes greater than or equal to `LimbCnt * CRYPTO_MPI_BITS_PER_LIMB`.

17.8.13 CRYPTO_MPI_LSB()

Description

Return the bit number of the least significant nonzero bit in the magnitude of Self. If Self is zero, CRYPTO_MPI_LSB() returns zero.

Prototype

```
unsigned CRYPTO_MPI_LSB(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Integer to size.

Return value

Bit number of least significant nonzero bit.

17.8.14 CRYPTO_MPI_MSB()

Description

Return the bit number of the most significant nonzero bit in the magnitude of Self.

Prototype

```
unsigned CRYPTO_MPI_MSB(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Integer to size.

Return value

< ost significant nonzero bit in the magnitude of Self.

Additional information

Although this is related to the `CRYPTO_MPI_BitCount()` function, the two are not trivially equivalent. For nonzero Self, the two are related by $\text{MSB}(x) = \text{BitCount}(x)-1$, but they differ when self is zero. In this case, both return zero.

17.8.15 CRYPTO_MPI_Unsigned()

Description

Return the least significant limb as an unsigned value, without respecting the sign of the argument.

Prototype

```
unsigned CRYPTO_MPI_Unsigned(const CRYPTO_MPI * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI.

Return value

Least-significant limb of MPI.

17.8.16 CRYPTO_MPI_LimbsRequired()

Description

Compute limbs required to hold value.

Prototype

```
unsigned CRYPTO_MPI_LimbsRequired(unsigned BitLen);
```

Parameters

Parameter	Description
BitLen	Number of bits required.

Return value

The number of limbs required to hold an MPI of `BitLen` bits.

17.9 Logical operations

Function	Description
CRYPTO_MPI_Xor()	Exclusive or.

17.9.1 CRYPTO_MPI_Xor()

Description

Exclusive or.

Prototype

```
int CRYPTO_MPI_Xor(      CRYPTO_MPI * pSelf,  
                      const CRYPTO_MPI * pValue);
```

Parameters

Parameter	Description
pSelf	Input #1 and output.
pValue	Input #2.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Exclusive or magnitude of inputs into pSelf.

17.10 Algorithms

Function	Description
CRYPTO_MPI_GCD()	Greatest common divisor.
CRYPTO_MPI_GCD_Binary()	Greatest common divisor, binary method.
CRYPTO_MPI_GCD_Euclid()	Greatest common divisor, Euclid method.
CRYPTO_MPI_GCD_Lehmer()	Greatest common divisor, Lehmer method.
CRYPTO_MPI_LCM()	Least common multiple.

17.10.1 CRYPTO_MPI_GCD()

Description

Greatest common divisor.

Prototype

```
int CRYPTO_MPI_GCD( CRYPTO_MPI * pSelf,
                     const CRYPTO_MPI * pX,
                     const CRYPTO_MPI * pY,
                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer assigned the GCD of X and Y.
pX	Pointer to MPI #1, X.
pY	Pointer to MPI #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0	Processing error.
≥ 0	Success.

17.10.2 CRYPTO_MPI_GCD_Binary()

Description

Greatest common divisor, binary method.

Prototype

```
int CRYPTO_MPI_GCD_Binary(      CRYPTO_MPI          * pSelf,
                                const CRYPTO_MPI        * pX,
                                const CRYPTO_MPI        * pY,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer assigned the GCD of X and Y.
pX	Pointer to MPI #1, X.
pY	Pointer to MPI #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.10.3 CRYPTO_MPI_GCD_Euclid()

Description

Greatest common divisor, Euclid method.

Prototype

```
int CRYPTO_MPI_GCD_Euclid(      CRYPTO_MPI          * pSelf,
                                const CRYPTO_MPI        * pX,
                                const CRYPTO_MPI        * pY,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer assigned the GCD of X and Y.
pX	Pointer to MPI #1, X.
pY	Pointer to MPI #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.10.4 CRYPTO_MPI_GCD_Lehmer()

Description

Greatest common divisor, Lehmer method.

Prototype

```
int CRYPTO_MPI_GCD_Lehmer(      CRYPTO_MPI          * pSelf,
                                const CRYPTO_MPI        * pX,
                                const CRYPTO_MPI        * pY,
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer assigned the GCD of X and Y.
pX	Pointer to MPI #1, X.
pY	Pointer to MPI #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.10.5 CRYPTO_MPI_LCM()

Description

Least common multiple.

Prototype

```
int CRYPTO_MPI_LCM( CRYPTO_MPI * pSelf,  
                     const CRYPTO_MPI * pX,  
                     const CRYPTO_MPI * pY,  
                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer assigned the LCM of X and Y.
pX	Input #1, X.
pY	Input #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

< 0 Processing error.
≥ 0 Success.

17.11 Random numbers

Function	Description
<code>CRYPTO_MPI_RandomBits()</code>	Generate random number.
<code>CRYPTO_MPI_Random()</code>	Generate random number.
<code>CRYPTO_MPI_NonzeroRandom()</code>	Generate nonzero random number.
<code>CRYPTO_MPI_NonzeroRandomEx()</code>	Nonzero random number, adjusted.

17.11.1 CRYPTO_MPI_NonzeroRandom()

Description

Generate nonzero random number.

Prototype

```
int CRYPTO_MPI_NonzeroRandom(      CRYPTO_MPI * pSelf,  
                                const CRYPTO_MPI * pMax);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the random value.
pMax	Pointer to MPI that contains the the (exclusive) maximum acceptable value.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Set `pSelf` to a random value V, 0 < V < Max.

17.11.2 CRYPTO_MPI_NonzeroRandomEx()

Description

Nonzero random number, adjusted.

Prototype

```
int CRYPTO_MPI_NonzeroRandomEx( CRYPTO_MPI * pSelf,
                                const CRYPTO_MPI * pMax,
                                int N);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the random value.
pMax	Pointer to MPI containing part of the (exclusive) maximum value acceptable.
N	Value to add to the MPI described by pMax to form the (exclusive) upper bound. It is acceptable for N to be negative to reduce the upper bound described by pMax.

Return value

< 0 Processing error.
 ≥ 0 Success.

Additional information

Set Self to a random value V, $0 < V < \text{Max} + N$. As N can be negative, the upper bound described by pMax can be reduced or increased as required.

17.11.3 CRYPTO_MPI_Random()

Description

Generate random number.

Prototype

```
int CRYPTO_MPI_Random(      CRYPTO_MPI * pSelf,  
                           const CRYPTO_MPI * pMax);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the random value.
pMax	Pointer to MPI containing the (exclusive) maximum acceptable value.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Set Self to a random value V, $0 \leq V < \text{Max}$.

17.11.4 CRYPTO_MPI_RandomBits()

Description

Generate random number.

Prototype

```
int CRYPTO_MPI_RandomBits(CRYPTO_MPI * pSelf,  
                           unsigned     BitLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the random bitstream.
BitLen	Number of bits in random number.

Return value

< 0 Processing error.
≥ 0 Success.

Additional information

Set `pSelf` to a random value V, $0 \leq V < 2^{\text{BitLen}}$.

17.12 Format conversion

Function	Description
<code>CRYPTO_MPI_FormatDecimal()</code>	Convert an MPI to a decimal representation, with leading sign, in Text.
<code>CRYPTO_MPI_FormatHex()</code>	Convert an MPI to hexadecimal representation, with leading sign, in Text.
<code>CRYPTO_MPI_Load()</code>	Read either a decimal, hexadecimal, or text representation of an MPI.
<code>CRYPTO_MPI_LoadBits()</code>	Read an array of bits from the array <code>aData[DataByteCnt]</code> where the most significant bit of stored data is bit 7 or <code>aData[DataByteCnt-1]</code> .
<code>CRYPTO_MPI_LoadBytes()</code>	Read an array of bytes, where each byte is taken as a base-256 digit, in network byte order.
<code>CRYPTO_MPI_LoadBytesLE()</code>	Read an array of bytes, where each byte is taken as a base-256 digit, in PC byte order.
<code>CRYPTO_MPI_LoadDecimal()</code>	Read a decimal representation of an MPI, with leading sign, and write it to Self.
<code>CRYPTO_MPI_LoadHex()</code>	Read a hexadecimal representation of an MPI, with leading sign but without "0x", and write it to Self.
<code>CRYPTO_MPI_LoadText()</code>	Read a string, where the ordinal value of each byte is taken as a base-256 digit to load, in network byte order.
<code>CRYPTO_MPI_StoreBytes()</code>	Store an integer in network byte order as an array of bytes.
<code>CRYPTO_MPI_StoreBytesLE()</code>	Store an integer in PC byte order as an array of bytes.

17.12.1 CRYPTO_MPI_FormatDecimal()

Description

Convert an MPI to a decimal representation, with leading sign, in Text.

Prototype

```
int CRYPTO_MPI_FormatDecimal(const CRYPTO_MPI * pSelf,  
                           char * sText,  
                           unsigned TextByteCnt,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Integer to print.
sText	Buffer to hold the result.
TextByteCnt	Size of the buffer in bytes.
pMem	Allocator to use for temporary storage.

Return value

Zero indicates value is formatted correctly into the output buffer and has a zero terminator. Nonzero indicates that the buffer is too small to hold the result and is unspecified on return; the value returned indicates the number of additional characters the buffer would need to be in order to satisfy the request without overflow.

17.12.2 CRYPTO_MPI_FormatHex()

Description

Convert an MPI to hexadecimal representation, with leading sign, in Text.

Prototype

```
int CRYPTO_MPI_FormatHex(const CRYPTO_MPI * pSelf,  
                           char      * sText,  
                           unsigned   TextLen);
```

Parameters

Parameter	Description
pSelf	Integer to format.
sText	Buffer to hold the result.
TextLen	Size of the buffer in bytes.

Return value

Zero indicates value is formatted correctly into the output buffer and has a zero terminator. Nonzero indicates that the buffer is too small to hold the result and is unspecified on return; the value returned indicates the number of additional characters the buffer would need to be in order to satisfy the request without overflow.

17.12.3 CRYPTO_MPI_Load()

Description

Read either a decimal, hexadecimal, or text representation of an MPI. Hexadecimal values start with a leading "0x" or "0X". Textual values start with a leading single or double quotation mark. If the string parsed without error, `pOK` is set nonzero; and if not, `pOK` is set to zero.

Prototype

```
int CRYPTO_MPI_Load(      CRYPTO_MPI * pSelf,
                        const char    * sText,
                        int           * pOK);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to load into.
<code>sText</code>	String to load from, null terminated.
<code>pOK</code>	If nonnull, set nonzero on success, set to zero on failure.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.12.4 CRYPTO_MPI_LoadBits()

Description

Read an array of bits from the array aData[DataByteCnt] where the most significant bit of stored data is bit 7 or aData[DataByteCnt-1].

Prototype

```
int CRYPTO_MPI_LoadBits(      CRYPTO_MPI * pSelf,
                           const U8          * pData,
                           unsigned        DataLen,
                           unsigned        LoadBitLen);
```

Parameters

Parameter	Description
pSelf	Integer to store into.
pData	Pointer to octet string to load from, in network byte order.
DataLen	Octet length of the octet string.
LoadBitLen	Number of bits to load.

Return value

< 0 Processing error.
≥ 0 Success.

17.12.5 CRYPTO_MPI_LoadBytes()

Description

Read an array of bytes, where each byte is taken as a base-256 digit, in network byte order. Network byte order is mandated by X.509, PKCS, and ASN.1 encoding.

Prototype

```
int CRYPTO_MPI_LoadBytes(      CRYPTO_MPI * pSelf,
                           const U8      * pData,
                           unsigned     DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the data.
pData	Pointer to object to read from, in network byte order.
DataLen	Octet length of the data object.

Return value

- < 0 Processing error.
 - ≥ 0 Success.
-
- < 0 Processing error.
 - ≥ 0 Success.

17.12.6 CRYPTO_MPI_LoadBytesLE()

Description

Read an array of bytes, where each byte is taken as a base-256 digit, in PC byte order.

Prototype

```
int CRYPTO_MPI_LoadBytesLE(      CRYPTO_MPI * pSelf,
                                const U8      * pData,
                                unsigned     DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that receives the data.
pData	Pointer to octet string to load from, in PC byte order.
DataLen	Octet length of the octet string.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.12.7 CRYPTO_MPI_LoadDecimal()

Description

Read a decimal representation of an MPI, with leading sign, and write it to `pSelf`. If any non-decimal digit is encountered, it is ignored and `pOK` is set to zero. If the string parsed without error, `pOK` is set to one.

Prototype

```
int CRYPTO_MPI_LoadDecimal(      CRYPTO_MPI * pSelf,
                                const char    * sText,
                                int           * pOK);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to load into.
<code>sText</code>	String to load from, null terminated.
<code>pOK</code>	If nonnull, set to nonzero on success, set to zero on failure.

Return value

< 0 Processing error.
≥ 0 Success.

17.12.8 CRYPTO_MPI_LoadHex()

Description

Read a hexadecimal representation of an MPI, with leading sign but without “`0x`”, and write it to Self. If any non-hexadecimal digit is encountered, it is ignored and `pOK` is set to zero. If the string parsed without error, `pOK` is set to one.

Prototype

```
int CRYPTO_MPI_LoadHex( CRYPTO_MPI * pSelf,  
                        const char      * sText,  
                        int             * pOK);
```

Parameters

Parameter	Description
<code>pSelf</code>	Integer to load into.
<code>sText</code>	String to load from, null terminated.
<code>pOK</code>	If nonnull, set to nonzero on success, set to zero on failure.

Return value

- < 0 Processing error.
- ≥ 0 Success.

17.12.9 CRYPTO_MPI_LoadText()

Description

Read a string, where the ordinal value of each byte is taken as a base-256 digit to load, in network byte order. So, for instance, the string "AB" would be loaded as the integer 0x4142, 16706 decimal.

Prototype

```
int CRYPTO_MPI_LoadText(      CRYPTO_MPI * pSelf,  
                           const char     * sText);
```

Parameters

Parameter	Description
pSelf	Integer to load into.
sText	String to load from, null terminated.

Return value

< 0	Processing error.
≥ 0	Success.

17.12.10 CRYPTO_MPI_StoreBytes()

Description

Store an integer in network byte order as an array of bytes. Network byte order is mandated by X.509, PKCS, and ASN.1 encoding. All bytes of the array are written, providing leading zero bytes if required.

Prototype

```
void CRYPTO_MPI_StoreBytes(const CRYPTO_MPI * pSelf,  
                           U8          * pData,  
                           unsigned     DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to store.
pData	Pointer to object that receives the octet string.
DataLen	Octet length of the stored octet string.

17.12.11 CRYPTO_MPI_StoreBytesLE()

Description

Store an integer in PC byte order as an array of bytes. All bytes of the array are written, providing trailing zero bytes if required.

Prototype

```
void CRYPTO_MPI_StoreBytesLE(const CRYPTO_MPI * pSelf,  
                           U8          * pData,  
                           unsigned     DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to store.
pData	Pointer to object that will receive the octet string.
DataLen	Octet length of the octet string.

17.13 Primes

Function	Description
Prime testing	
<code>CRYPTO_MPI_IsCoprime()</code>	Are MPIs coprime?
<code>CRYPTO_MPI_IsFermatProbablePrime()</code>	Answer whether P is a Fermat probable prime.
<code>CRYPTO_MPI_IsMRProbablePrime()</code>	Answer whether P is a probable prime by running Fermat and Miller-Rabin probable primality tests.
<code>CRYPTO_MPI_IsMRProbablePrimeEx()</code>	Answer whether P is a Miller-Rabin probable prime after running multiple Miller-Rabin trials.
<code>CRYPTO_MPI_IsProbableSafePrime()</code>	Answer whether Self is a probable strong prime.
<code>CRYPTO_MPI_IsProvableSmallPrime()</code>	Answer whether Self is a known prime, by trial division.
<code>CRYPTO_MPI_P1363_CalcMRTrials()</code>	Answer the minimum number of Miller-Rabin trials to achieve a probability of error less than 2^{-100} for a random BitCnt-bit integer.
<code>CRYPTO_MPI_QuerySmallPrimeFactor()</code>	Answer whether P is a known composite by dividing by small prime factors.
Prime generation	
<code>CRYPTO_PRIME_FindPrime()</code>	Find a probable prime, Prime, of at least BitCnt bits.
<code>CRYPTO_PRIME_FindPrimeFrom()</code>	Find the next probable prime greater than or equal to Prime.
<code>CRYPTO_PRIME_FindCoprime()</code>	Find a prime [pmin, pmax] such that p-1 is relatively prime to an odd positive integer f.

17.13.1 CRYPTO_MPI_IsCoprime()

Description

Are MPIs coprime?

Prototype

```
int CRYPTO_MPI_IsCoprime(const CRYPTO_MPI * pX,  
                           const CRYPTO_MPI * pY,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pX	Pointer to input #1, X.
pY	Pointer to input #2, Y.
pMem	Pointer to allocator that provides temporary storage.

Return value

- > 0 Parameters are coprime.
- = 0 Parameters share a common factor and are not coprime.
- < 0 Processing error.

Additional information

X and Y are coprime if they share no common factor. For instance, 27 and 34 are coprime as the prime factorizations $27=3\times 9$ and $34=2\times 17$ where there is no common factor.

17.13.2 CRYPTO_MPI_IsFermatProbablePrime()

Description

Answer whether P is a Fermat probable prime.

Prototype

```
int CRYPTO_MPI_IsFermatProbablePrime(const CRYPTO_MPI * pP,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pP	Pointer to MPI to test.
pMem	Allocator to use for temporary storage.

Return value

- > 0 Proven composite 0 - Not proven composite
- < 0 Processing error

17.13.3 CRYPTO_MPI_IsMRProbablePrime()

Description

Answer whether P is a probable prime by running Fermat and Miller-Rabin probable primality tests.

Prototype

```
int CRYPTO_MPI_IsMRProbablePrime(const CRYPTO_MPI           * pSelf,  
                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.
pMem	Allocator to use for temporary storage.

Return value

- > 0 Probable prime
- = 0 Known composite, not a prime
- < 0 Processing error

17.13.4 CRYPTO_MPI_IsMRProbablePrimeEx()

Description

Answer whether P is a Miller-Rabin probable prime after running multiple Miller-Rabin trials.

Prototype

```
int CRYPTO_MPI_IsMRProbablePrimeEx(const CRYPTO_MPI * pSelf,  
                                    unsigned Trials,  
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Candidate prime to test.
Trials	Number of trials to run.
pMem	Allocator to use for temporary storage.

Return value

- > 0 Miller-Rabin probable prime
- = 0 Known composite, not a prime
- < 0 Error status indication

Additional information

The number of trials you specify for the MR test depends upon the source of the prime number, i.e. whether it is generated or not. Consult the appropriate standard before use!

17.13.5 CRYPTO_MPI_IsProbableSafePrime()

Description

Answer whether Self is a probable strong prime. A prime of the form $2p+1$ is a safe prime if p is also a prime.

Prototype

```
int CRYPTO_MPI_IsProbableSafePrime(const CRYPTO_MPI * pSelf,  
CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.
pMem	Allocator to use for temporary storage.

Return value

- > 0 A probable safe prime
- = 0 Not a probable safe prime
- < 0 Processing error

17.13.6 CRYPTO_MPI_IsProvableSmallPrime()

Description

Answer whether `Self` is a known prime, by trial division.

Prototype

```
int CRYPTO_MPI_IsProvableSmallPrime(U32 Self);
```

Parameters

Parameter	Description
<code>Self</code>	Value to test for primality.

Return value

> 0 Proven small prime < 2^{32} 0 - Not proven small prime

17.13.7 CRYPTO_MPI_P1363_CalcMRTrials()

Description

Answer the minimum number of Miller-Rabin trials to achieve a probability of error less than 2^{-100} for a random `BitCnt`-bit integer.

Prototype

```
unsigned CRYPTO_MPI_P1363_CalcMRTrials(unsigned BitCnt);
```

Parameters

Parameter	Description
<code>BitCnt</code>	Number of bits in the value to test.

Return value

Minimun number of Miller-Rabin trials.

17.13.8 CRYPTO_MPI_QuerySmallPrimeFactor()

Description

Answer whether P is a known composite by dividing by small prime factors.

Prototype

```
int CRYPTO_MPI_QuerySmallPrimeFactor(const CRYPTO_MPI * pSelf,  
                                     CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI to test.
pMem	Allocator to use for temporary storage.

Return value

- > 0 Proven composite 0 - Not proven composite
- < 0 Processing error

17.13.9 CRYPTO_PRIME_FindPrime()

Description

Find a probable prime, Prime, of at least `BitCnt` bits. If `q` is nonnull and nonzero, ensure that `Prime-1` is relatively prime to `q`.

Prototype

```
int CRYPTO_PRIME_FindPrime( CRYPTO_MPI * pPrime,  
                           unsigned BitCnt,  
                           const CRYPTO_MPI * pQ,  
                           CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
<code>pPrime</code>	Pointer to MPI that receives the prime, if successful.
<code>BitCnt</code>	Width of requested prime, in bits.
<code>pQ</code>	Pointer to MPI that generated prime must be coprime to.
<code>pMem</code>	Pointer to memory allocation API for temporary storage.

Return value

> 0 Prime found
< 0 Error status indication

Additional information

Zero is never returned from this function

17.13.10 CRYPTO_PRIME_FindPrimeFrom()

Description

Find the next probable prime greater than or equal to Prime. If q is nonnull and nonzero, ensure that Prime-1 is relatively prime to q.

Prototype

```
int CRYPTO_PRIME_FindPrimeFrom( CRYPTO_MPI * pPrime,  
                                const CRYPTO_MPI * pQ,  
                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrime	in Pointer to MPI to start search from. out Pointer to MPI that receives the prime, if successful.
pQ	Pointer to MPI that pPrime must be coprime to.
pMem	Pointer to memory allocation API for temporary storage.

Return value

- > 0 Prime found
- < 0 Error status indication

Additional information

Zero is never returned from this function

17.13.11 CRYPTO_PRIME_FindCoprime()

Description

Find a prime [pmin, pmax] such that p-1 is relatively prime to an odd positive integer f.

Prototype

```
int CRYPTO_PRIME_FindCoprime(      CRYPTO_MPI      * pPrime,
                                    const CRYPTO_MPI   * pMin,
                                    const CRYPTO_MPI   * pMax,
                                    const CRYPTO_MPI   * pF,
                                    CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pPrime	Pointer to MPI that receives the prime, if successful.
pMin	Pointer to MPI that contains the minimum acceptable value for the prime.
pMax	Pointer to MPI that contains the maximum acceptable value for the prime.
pF	Pointer to MPI that the generated prime minus one must be coprime to.
pMem	Pointer to memory allocation API for temporary storage.

Return value

> 0 Prime found
 < 0 Error status indication

Additional information

Zero is never returned from this function

Chapter 18

Cryptographic message syntax

18.1 CCM and GCM

Function	Description
CRYPTO_CMS_Rd_CCMPParameters()	Read CMS CCMPParameters.
CRYPTO_CMS_Rd_GCMPParameters()	Read CMS GCMPParameters.

18.1.1 CRYPTO_CMS_Rd_CCMPParameters()

Description

Read CMS CCMPParameters.

Prototype

```
int CRYPTO_CMS_Rd_CCMPParameters(CRYPTO_TLV * pTLV,  
                                  CRYPTO_CMS_GCM_CCM_PARAMETERS * pParas);
```

Parameters

Parameter	Description
pTLV	Pointer to TLV containing the parameters.
pParas	Pointer to object that receives the parsed parameters.

Return value

- ≥ 0 Success.
- < 0 Processing error.

18.1.2 CRYPTO_CMS_Rd_GCMParameters()

Description

Read CMS GCMParameters.

Prototype

```
int CRYPTO_CMS_Rd_GCMParameters(CRYPTO_TLV * pTLV,  
                                 CRYPTO_CMS_GCM_CCM_PARAMETERS * pParas);
```

Parameters

Parameter	Description
pTLV	Pointer to TLV containing the parameters.
pParas	Pointer to object that receives the parsed parameters.

Return value

- ≥ 0 Success.
- < 0 Processing error.

Chapter 19

Hardware acceleration

19.1 EFM32 CRYPTO coprocessor (Add-on)

The EFM32 cryptographic coprocessor (CRYPTO) is presented as a memory-mapped peripheral.

emCrypt has specialized hardware-assisted hashing support for the following cryptographic algorithms using the CRYPTO coprocessor:

- SHA-1

19.1.1 Installing CRYPTO hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, NULL);
}
```

19.1.2 EFM32 cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 covering hashing and RSA operations. See *Crypto-OS integration* on page 31 for further details.

If you wish to reduce power consumption, it is possible to enable and disable the crypto unit when `CRYPTO_OS_Claim()` is called and disable them `CRYPTO_OS_Unclaim()` is called (for cryptographic unit #0).

19.1.3 Modular exponentiation API

Function	Description
Windowing, Montgomery reduction	
<code>CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 2-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 3-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 4-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 5-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 6-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 2-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 3-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 4-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 5-bit window.
<code>CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPT0()</code>	Modular exponentiation, Montgomery reduction, 6-bit window.

19.1.3.1 CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.3.2 CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

19.1.3.3 CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.3.4 CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.3.5 CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.3.6 CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.3.7 CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

19.1.3.8 CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

19.1.3.9 CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0 Processing error
≥ 0 Success

19.1.3.10 CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO
    ( CRYPTO_MPI * pSelf,
      const CRYPTO_MPI * pExponent,
      const CRYPTO_MPI * pModulus,
      CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0	Processing error
≥ 0	Success

19.1.4 Performance

19.1.4.1 SHA-1

Output from the benchmark CRYPTO_Bench_SHA1 is shown below.

```
(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG      www.segger.com
SHA-1 Benchmark V2.00 compiled May 24 2017 12:06:22

Compiler: clang 4.0.0 (tags/RELEASE_400/final)
System: Processor speed          = 19.000 MHz
Config: CRYPTO_CONFIG_SHA1_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA1_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-1     |    0.76   |
| SHA-1 (HW) |    6.77   |
+-----+-----+

Benchmark complete
```

19.1.5 Sample EFM32 setup

The following is the cryptographic setup for the Silicon Labs Pearl and Jade Gecko devices:

```
*****
*          (c) SEGGER Microcontroller GmbH & Co. KG
*          The Embedded Experts
*          www.segger.com
*****  
----- END-OF-HEADER -----  
  
File      : CRYPTO_X_Config_EFM32.c  
Purpose   : Configure CRYPTO for EFM32 Pearl and Jade Geckos.  
  
*/  
  
*****  
*  
*      #include Section  
*  
*****  
*/  
  
#include "CRYPTO.h"  
  
*****  
*  
*      Public code  
*  
*****  
*/  
  
*****  
*      CRYPTO_X_Panic()  
*  
*      Function description  
*      Hang when something unexpected happens.  
*/  
void CRYPTO_X_Panic(void) {  
    for (;;) {  
        /* Hang */  
    }  
}  
  
*****  
*  
*      CRYPTO_X_Config()  
*  
*      Function description  
*      Configure hardware assist for CRYPTO component.  
*/  
void CRYPTO_X_Config(void) {  
    volatile U32 *HFBUSCLKEN0;  
    //  
    CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,           NULL);  
    CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, NULL);  
    CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,        NULL);  
    CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,        NULL);  
    CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES,         NULL);  
    CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES,        NULL);  
    CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,       NULL);  
    CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,    NULL);  
    //  
    // Clock CRYPTO peripheral.  
    //
```

```
HFBUSCLKEN0 = (void *)0x400E40B0UL;
*HFBUSCLKEN0 |= 1UL << 1; // Turn on clock to CRYPTO unit
}

//********************************************************************* End of file *****/
```

19.2 Kinetis CAU coprocessor (Add-on)

The Kinetis Cryptographic Acceleration Unit (CAU) is a primitive accelerator presented as a memory-mapped peripheral.

emCrypt has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the CAU:

- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.
- MD5
- SHA-1
- SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software managing the cipher mode.

19.2.1 Installing CAU hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_Kinetis_CAU;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_MD5_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA1_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA224_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_Kinetis_CAU;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_MD5_Install  (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,      NULL);
    CRYPTO_SHA1_Install (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,      NULL);
    CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_Kinetis_CAU,  NULL);
    CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_Kinetis_CAU,  NULL);
    CRYPTO_AES_Install  (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,    NULL);
    CRYPTO_TDES_Install (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU,   NULL);
}
```

Note

Whilst there is an MD5 accelerator, hardware-assisted MD5 is slower than a pure software implementation of MD5 using Thumb-2 so we recommend that you do not install the MD5 accelerator.

19.2.2 Kinetis cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 31 for further details.

19.2.3 Sample Kinetis setup

The following is the cryptographic setup for the SEGGER emPower board based on the Kinetis K66 device.

```
/*
 *      (c) SEGGER Microcontroller GmbH & Co. KG
 *      The Embedded Experts
 *      www.segger.com
 */

----- END-OF-HEADER -----
File      : CRYPTO_X_Config_K66.c
Purpose   : Configure CRYPTO for K66 devices.

*/
/* #include Section */
*/
#include "CRYPTO.h"

/*
 *      Public code
 */
/*
 *      CRYPTO_X_Panic()
 *
 *      Function description
 *      Hang when something unexpected happens.
 */
void CRYPTO_X_Panic(void) {
    for (;;) {
        /* Hang */
    }
}

/*
 *      CRYPTO_X_Config()
 *
 *      Function description
 *      Configure hardware assist for CRYPTO component.
 */
void CRYPTO_X_Config(void) {
    volatile U32 *pReg;
    //
    // Install hardware assistance.
    //
    CRYPTO_MD5_Install     (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,      NULL);
    CRYPTO_SHA1_Install    (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,      NULL);
    CRYPTO_SHA224_Install  (&CRYPTO_HASH_SHA224_HW_Kinetis_CAU, NULL);
    CRYPTO_SHA256_Install  (&CRYPTO_HASH_SHA256_HW_Kinetis_CAU, NULL);
    CRYPTO_AES_Install     (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,  NULL);
    CRYPTO_TDES_Install    (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, NULL);
    //
    // Software ciphers.
    //
}
```

```
CRYPTO_CAST_Install      (&CRYPTO_CIPHER_CAST_SW,      NULL);
CRYPTO_SEED_Install      (&CRYPTO_CIPHER_SEED_SW,      NULL);
CRYPTO_ARIA_Install      (&CRYPTO_CIPHER_ARIA_SW,      NULL);
CRYPTO_CAMELLIA_Install  (&CRYPTO_CIPHER_CAMELLIA_SW,  NULL);
CRYPTO_BLOWFISH_Install  (&CRYPTO_CIPHER_BLOWFISH_SW,  NULL);
CRYPTO_TWOFISH_Install   (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
//
// Software hashing.
//
CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,      NULL);
CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,  NULL);
//
// Turn on clocks to RNGA, bit 0 of SIM_SCGC3, and install RNG.
//
pReg = (void *)0x40048030;
*pReg |= 1;
//
// Install Hash_DRBG-SHA-256 with RNGA entropy.
//
CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_Kinetis_RNGA);
//
// Install small modular exponentiation functions.
//
CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

***** End of file *****
```

19.3 LPC18S and LPC43S AES ROM (Add-on)

The LPC18Sxx and LPC43Sxx microcontrollers provide an AES-128 hardware accelerator. The capabilities of this accelerator are exposed through a ROM-based API which insulates the programmer from changes to or variants of the underlying accelerator hardware.

emCrypt has specialized hardware-assisted AES ciphering for the following cryptographic algorithms:

- AES-128 in ECB and CBC modes.

All other AES-128 cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software managing the cipher mode. All ciphering with AES-192 and AES-256 falls back to using a pure software AES kernel.

19.3.1 Installing LPC ROM hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_LPC_ROM;
```

If all you require is AES-128, you can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM, 0);
}
```

However, if you require AES-192 or AES-256 in addition to AES-128, you must install a software fallback for these key sizes:

```
void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM,
                      &CRYPTO_CIPHER_AES_SW);
}
```

19.3.2 LPC cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 31 for further details.

19.3.3 Sample LPS18S setup

The following is the cryptographic setup for the NXP LPCXpresso18S37 board:

```
*****
*          (c) SEGGER Microcontroller GmbH & Co. KG      *
*          The Embedded Experts                         *
*          www.segger.com                           *
*****  

----- END-OF-HEADER -----  

File      : CRYPTO_X_Config_LPC18S37.c
Purpose   : Configure CRYPTO for LPC18S37 devices.  

*/  

/*****  

*  

*      #include Section  

*  

*****  

*/  

#include "CRYPTO.h"  

/*****  

*  

*      Public code  

*  

*****  

*/  

/*****  

*  

*      CRYPTO_X_Panic()  

*  

*      Function description  

*      Hang when something unexpected happens.  

*/  

void CRYPTO_X_Panic(void) {
    for (;;) {
        /* Hang */
    }
}  

/*****  

*  

*      CRYPTO_X_Config()  

*  

*      Function description  

*      Configure hardware assist for CRYPTO component.  

*/  

void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_LPC_ROM, &CRYPTO_CIPHER_AES_SW);
    CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,      0);
    CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,       0);
    CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,      0);
    CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,    0);
    CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,    0);
    CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, 0);
}  

***** End of file *****
```

19.4 iMX RT10xx data coprocessor (Add-on)

The iMX RT10xx Data Coprocessor (DCP) is a programmable cryptographic accelerator presented as a memory-mapped peripheral.

emCrypt has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the DCP:

- AES-128 in ECB and CBC modes.
- SHA-1
- SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software managing the cipher mode.

19.4.1 Installing iMX RT10xx hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API    CRYPTO_HASH_SHA1_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API    CRYPTO_HASH_SHA256_HW_RT10xx_DCP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,
                          &CRYPTO_HASH_SHA1_SW);
    CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_RT10xx_DCP,
                         &CRYPTO_HASH_SHA256_SW);
    CRYPTO_AES_Install   (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,
                          &CRYPTO_CIPHER_AES_SW);
    //
    // Install Hash_DRBG-SHA-256 with TRNG entropy.
    //
    CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                         &CRYPTO_RNG_HW_RT10xx_TRNG);
}
```

19.4.2 RT10xx cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 31 for further details.

19.4.3 Sample Kinetis setup

The following is the cryptographic setup for the SEGGER RT1051 Trace Reference board.

```
*****
*          (c) SEGGER Microcontroller GmbH
*          The Embedded Experts
*          www.segger.com
*****  
----- END-OF-HEADER -----  
  
File      : CRYPTO_X_Config_RT10xx.c  
Purpose   : Configure CRYPTO for iMX RT10xx devices.  
  
*/  
  
*****  
*  
*      #include Section  
*  
*****  
*/  
  
#include "CRYPTO.h"  
  
*****  
*  
*      Public code  
*  
*****  
*/  
  
*****  
*      CRYPTO_X_Panic()  
*  
*      Function description  
*      Hang when something unexpected happens.  
*/  
void CRYPTO_X_Panic(void) {  
    for (;;) {  
        /* Hang */  
    }  
}  
  
*****  
*  
*      CRYPTO_X_Config()  
*  
*      Function description  
*      Configure hardware assist for CRYPTO component.  
*/  
void CRYPTO_X_Config(void) {  
    //  
    // Install hardware assistance.  
    //  
    CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,  
    &CRYPTO_HASH_SHA1_SW);  
    CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_HW_RT10xx_DCP, &CRYPTO_HASH_SHA256_SW);  
    CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,  
    &CRYPTO_CIPHER_AES_SW);  
    //  
    // Software ciphers.  
    //  
    CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,      NULL);
```

```
CRYPTO_CAST_Install      (&CRYPTO_CIPHER_CAST_SW,      NULL) ;
CRYPTO_SEED_Install      (&CRYPTO_CIPHER_SEED_SW,      NULL) ;
CRYPTO_ARIA_Install      (&CRYPTO_CIPHER_ARIA_SW,      NULL) ;
CRYPTO_CAMELLIA_Install  (&CRYPTO_CIPHER_CAMELLIA_SW,  NULL) ;
CRYPTO_BLOWFISH_Install  (&CRYPTO_CIPHER_BLOWFISH_SW,  NULL) ;
CRYPTO_TWOFISH_Install   (&CRYPTO_CIPHER_TWOFISH_SW,  NULL) ;
//
// Software hashing.
//
CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,      NULL) ;
CRYPTO_SHA1_Install       (&CRYPTO_HASH_SHA1_SW,      NULL) ;
CRYPTO_SHA224_Install    (&CRYPTO_HASH_SHA224_SW,     NULL) ;
CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,     NULL) ;
CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,     NULL) ;
CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,  NULL) ;
//
// Install Hash_DRBG-SHA-256 with TRNG entropy.
//
CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_RT10xx_TRNG) ;
//
// Install small modular exponentiation functions.
//
CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast) ;
CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast) ;
}

***** End of file *****
```

19.5 STM32 AES coprocessor (Add-on)

The STM32 AES hardware accelerator (AES) is a hardware accelerator presented as a memory-mapped peripheral that accelerates AES-128 and AES-256 encryption and decryption. The AES accelerator is present on selected STM32L4 devices.

emCrypt has support for the following cryptographic algorithms using the AES hardware accelerator:

- AES-128 and AES-256 in ECB and CBC modes.

19.5.1 Installing AES hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_AES;
```

You can install hardware support for AES-128 and AES-192 *only* using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES, NULL);
}
```

If you require AES-192 support, you must install a software fallback that is used when ciphering with a 192-bit key:

```
void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES,
                        &CRYPTO_CIPHER_AES_SW);
}
```

19.5.2 Enabling the AES coprocessor

You must enable clocks and reset the AES peripheral before reading or writing its registers. For the STM32L4A6 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//  
pReg = (volatile U32 *)0x4002104C; // RCC_AHB2ENR  
*pReg |= 1U << 4; // RCC_AHB2ENR.AESEN=1  
pReg = (volatile U32 *)0x4002102C; // RCC_AHB2RSTR  
*pReg |= 1U << 16; // RCC_AHB2RSTR.AESRST=1  
*pReg &= ~(1U << 16); // RCC_AHB2RSTR.AESRST=0
```

19.5.3 STM32 cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 31 for further details.

19.6 STM32 CRYP coprocessor (Add-on)

The STM32 cryptographic processor (CRYP) is a capable hardware accelerator presented as a memory-mapped peripheral that accelerates AES and TDES encryption and decryption. There are two variants of the CRYP processor with different capabilities present on the following family members:

- STM32F41x CRYP, hereafter referred to as the *standard CRYP processor*, and
- STM32F43x/F47x CRYP, hereafter referred to as the *enhanced CRYP processor*.

emCrypt has support for the following cryptographic algorithms using both CRYP variants:

- DES in ECB and CBC modes.
- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.

For the enhanced CRYP processor, direct acceleration is provided for:

- AES-128, AES-192, and AES-256 in CCM(12,4) and GCM(12,4) modes.

For the standard CRYP processor, acceleration is provided for:

- AES-128, AES-192, and AES-256 ciphering with GCM and CCM in software.

For CCM and GCM modes, the CRYP processor supports only fixed 16-byte authentication tags and 12-byte IVs with 4-byte counters. Therefore, AES-CCM acceleration is not immediately suitable for authenticated encryption in SSH as SSH requires zero-length IVs with 16-byte counters.

19.6.1 Installing CRYP hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_CRYP;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_STM32_CRYP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_STM32_CRYP);
    CRYPTO_TDES_Install(&CRYPTO_CIPHER_TDES_HW_STM32_CRYP);
}
```

19.6.2 Enabling the CRYP coprocessor

You must enable clocks and reset the CRYP peripheral before reading or writing its registers. For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//  

pReg = (volatile U32 *)0x40023834; // RCC_AHB2ENR  

*pReg |= 1U << 4; // RCC_AHB2ENR.CRYPEN=1  

pReg = (volatile U32 *)0x40023814; // RCC_AHB2RSTR  

*pReg |= 1U << 4; // RCC_AHB2RSTR.CRYPRST=1  

*pReg &= ~(1U << 4); // RCC_AHB2RSTR.CRYPRST=0
```

19.6.3 STM32 cryptographic units

The emCrypt implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 31 for further details.

19.6.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```
/*
 *      (c) SEGGER Microcontroller GmbH & Co. KG
 *      The Embedded Experts
 *      www.segger.com
 */

----- END-OF-HEADER -----


File      : CRYPTO_X_Config_STM32F75x.c
Purpose   : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/



/*
 *      #include Section
 *
*****



#include "CRYPTO.h"

/*
 *      Public code
 *
*****



*/
void CRYPTO_X_Panic(void) {
    for (;;) {
        /* Hang */
    }
}

/*
 *      CRYPTO_X_Config()
 *
 *      Function description
 *      Hang when something unexpected happens.
 */
void CRYPTO_X_Config(void) {
    volatile U32 *pReg;
    //
    // Turn on clocks to the CRYP accelerator and reset it.
    //
    pReg = (volatile U32 *)0x40023834;    // RCC_AHB2ENR
    *pReg |= 1u << 4;                  // RCC_AHB2ENR.CRYPTEN=1
    pReg = (volatile U32 *)0x40023814;    // RCC_AHB2RSTR
    *pReg |= 1u << 4;                  // RCC_AHB2RSTR.CRYPRST=1
    *pReg &= ~(1u << 4);              // RCC_AHB2RSTR.CRYPRST=0
    //
    // Install cipher hardware assistance.
    //
}
```

```

CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
// 
// Turn on clocks to the HASH accelerator and reset it.
//
pReg = (volatile U32 *)0x40023834;    // RCC_AHB2ENR
*pReg |= 1u << 5;                  // RCC_AHB2ENR.HASHEN=1
pReg = (volatile U32 *)0x40023814;    // RCC_AHB2RSTR
*pReg |= 1u << 5;                  // RCC_AHB2RSTR.HASRST=1
*pReg &= ~(1u << 5);              // RCC_AHB2RSTR.HASRST=0
//
// Install hardware hashing with software fallback (required).
//
CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
&CRYPTO_HASH_MD5_SW);
CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
&CRYPTO_HASH_SHA1_SW);
CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
//
// Software hashing.
//
CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,        NULL);
CRYPTO_SEED_Install      (&CRYPTO_CIPHER_SEED_SW,       NULL);
CRYPTO_ARIA_Install      (&CRYPTO_CIPHER_ARIA_SW,      NULL);
CRYPTO_CAMELLIA_Install  (&CRYPTO_CIPHER_CAMELLIA_SW,  NULL);
//
// Turn on clocks to the RNG and reset it.
//
pReg = (volatile U32 *)0x40023834;    // RCC_AHB2ENR
*pReg |= 1u << 6;                  // RCC_AHB2ENR.RNGEN=1
pReg = (volatile U32 *)0x40023814;    // RCC_AHB2RSTR
*pReg |= 1u << 6;                  // RCC_AHB2RSTR.RNGRST=1
*pReg &= ~(1u << 6);              // RCC_AHB2RSTR.RNGRST=0
//
// Random number generator.
//
CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG,  &CRYPTO_RNG_HW_STM32_RNG);
//
// Install small modular exponentiation functions.
//
CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

***** End of file *****/

```

19.7 STM32 HASH coprocessor (Add-on)

The STM32 hash coprocessor (HASH) is a hardware accelerator presented as a memory-mapped peripheral that accelerates calculation of MD5, SHA-1, SHA-224 and SHA-256 message digests.

emCrypt has HASH accelerator support for the following cryptographic algorithms:

- MD5 message digest.
- SHA-1 message digest.
- SHA-224 and SHA-256 message digest.

19.7.1 Installing HASH hardware support

The following interfaces are provided:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_MD5_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA224_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_HW_STM32_HASH;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
    CRYPTO_MD5_Install(&CRYPTO_HASH_MD5_HW_STM32_HASH);
    CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_HW_STM32_HASH);
    CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_STM32_HASH);
    CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_STM32_HASH);
}
```

19.7.2 Enabling the HASH coprocessor

You must enable clocks and reset the HASH peripheral before reading or writing its registers.

For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//  

pReg = (volatile U32 *)0x40023834; // RCC_AHB2ENR  

*pReg |= 1u << 5; // RCC_AHB2ENR.HASHEN=1  

pReg = (volatile U32 *)0x40023814; // RCC_AHB2RSTR  

*pReg |= 1u << 5; // RCC_AHB2RSTR.HASRST=1  

*pReg &= ~(1u << 5); // RCC_AHB2RSTR.HASRST=0
```

For the STM32L4 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *RCC_AHB2RSTR = (U32 *)0x4002102C;  

volatile U32 *RCC_AHB2ENR = (U32 *)0x4002104C;  

//  

*RCC_AHB2ENR |= 1<<17;  

*RCC_AHB2RSTR |= 1<<17;  

*RCC_AHB2RSTR &= ~(1<<17);
```

19.7.3 STM32 cryptographic units

The emCrypt implementation of hardware assistance requires two cryptographic units with indexes #0 and #1 that cover ciphering (unit #0) and hashing (unit #1). See *CRYPTO-OS integration* on page 31 for further details.

19.7.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```
/*
 *      (c) SEGGER Microcontroller GmbH & Co. KG
 *      The Embedded Experts
 *      www.segger.com
 */

----- END-OF-HEADER -----


File      : CRYPTO_X_Config_STM32F75x.c
Purpose   : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/



/*
 *      #include Section
 *
*****



#include "CRYPTO.h"

/*
 *      Public code
 *
*****



*/
void CRYPTO_X_Panic(void) {
    for (;;) {
        /* Hang */
    }
}

/*
 *      CRYPTO_X_Config()
 *
 *      Function description
 *      Hang when something unexpected happens.
 */
void CRYPTO_X_Config(void) {
    volatile U32 *pReg;
    //
    // Turn on clocks to the CRYP accelerator and reset it.
    //
    pReg = (volatile U32 *)0x40023834;    // RCC_AHB2ENR
    *pReg |= 1u << 4;                  // RCC_AHB2ENR.CRYPTEN=1
    pReg = (volatile U32 *)0x40023814;    // RCC_AHB2RSTR
    *pReg |= 1u << 4;                  // RCC_AHB2RSTR.CRYPRST=1
    *pReg &= ~(1u << 4);              // RCC_AHB2RSTR.CRYPRST=0
    //
    // Install cipher hardware assistance.
    //
}
```

```

CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
// 
// Turn on clocks to the HASH accelerator and reset it.
//
pReg = (volatile U32 *)0x40023834; // RCC_AHB2ENR
*pReg |= 1u << 5;                // RCC_AHB2ENR.HASHEN=1
pReg = (volatile U32 *)0x40023814; // RCC_AHB2RSTR
*pReg |= 1u << 5;                // RCC_AHB2RSTR.HASRST=1
*pReg &= ~(1u << 5);            // RCC_AHB2RSTR.HASRST=0
//
// Install hardware hashing with software fallback (required).
//
CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
&CRYPTO_HASH_MD5_SW);
CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
&CRYPTO_HASH_SHA1_SW);
CRYPTO_SHA224_Install    (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
//
// Software hashing.
//
CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,        NULL);
CRYPTO_SEED_Install      (&CRYPTO_CIPHER_SEED_SW,       NULL);
CRYPTO_ARIA_Install      (&CRYPTO_CIPHER_ARIA_SW,       NULL);
CRYPTO_CAMELLIA_Install  (&CRYPTO_CIPHER_CAMELLIA_SW,    NULL);
//
// Turn on clocks to the RNG and reset it.
//
pReg = (volatile U32 *)0x40023834; // RCC_AHB2ENR
*pReg |= 1u << 6;                // RCC_AHB2ENR.RNGEN=1
pReg = (volatile U32 *)0x40023814; // RCC_AHB2RSTR
*pReg |= 1u << 6;                // RCC_AHB2RSTR.RNGRST=1
*pReg &= ~(1u << 6);            // RCC_AHB2RSTR.RNGRST=0
//
// Random number generator.
//
CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG, &CRYPTO_RNG_HW_STM32_RNG);
//
// Install small modular exponentiation functions.
//
CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

***** End of file *****/

```

Chapter 20

Utilities

20.1 Buffer manipulation

20.1.1 Type-safe API

The following table lists the buffer API functions.

Function	Description
Management functions	
CRYPTO_BUFFER_Init()	Initialize write buffer.
CRYPTO_BUFFER_Reset()	Reset buffer to empty state.
Inquiry functions	
CRYPTO_BUFFER_SpaceLeft()	Inquire number of bytes that can be written.
CRYPTO_BUFFER_Overflow()	Inquire the number of bytes that were not written.
CRYPTO_BUFFER_Status()	Return status indicating buffer overflow.
CRYPTO_BUFFER_Data()	Return pointer to start of buffer.
Cursor functions	
CRYPTO_BUFFER_Cursor()	Return front position of buffer.
CRYPTO_BUFFER_CursorDistance()	Return cursor distance beyond mark.
CRYPTO_BUFFER_CursorIndex()	Return the cursor position.
CRYPTO_BUFFER_SetCursor()	Set the cursor index.
CRYPTO_BUFFER_SetCursorIndex()	Set the cursor index.
CRYPTO_BUFFER_Skip()	Advance write pointer in buffer.
Immediate write functions	
CRYPTO_BUFFER_Copy()	Copy object to buffer.
CRYPTO_BUFFER_Wr()	Write object to buffer.
CRYPTO_BUFFER_WrLogical()	Write a block to the write buffer and apply logic operation.
CRYPTO_BUFFER_WrStr()	Write string to buffer.
CRYPTO_BUFFER_WrStrLn()	Write string and end-of-line to buffer.
CRYPTO_BUFFER_WrU8()	Write a single byte to the buffer.
CRYPTO_BUFFER_WrU16LE()	Write a 16-bit unsigned integer in PC byte order to the write buffer.
CRYPTO_BUFFER_WrU16BE()	Write a 16-bit unsigned integer in network byte order to the write buffer.
CRYPTO_BUFFER_WrU24LE()	Write a 24-bit unsigned integer in PC byte order to the write buffer.
CRYPTO_BUFFER_WrU24BE()	Write a 24-bit unsigned integer in network byte order to the write buffer.
CRYPTO_BUFFER_WrU32LE()	Write a 32-bit unsigned integer in PC byte order to the write buffer.
CRYPTO_BUFFER_WrU32BE()	Write a 32-bit unsigned integer in network byte order to the write buffer.
CRYPTO_BUFFER_WrU64LE()	Write a 64-bit unsigned integer in PC byte order to the write buffer.
CRYPTO_BUFFER_WrU64BE()	Write a 64-bit unsigned integer in network byte order to the write buffer.

Function	Description
<code>CRYPTO_BUFFER_WrCntU32BE()</code>	Write counted string.
<code>CRYPTO_BUFFER_WrMultiU8()</code>	Write multiple identical bytes.
<code>CRYPTO_BUFFER_MPI_WrCounted()</code>	Write a length-counted MPI to the buffer.
<code>CRYPTO_BUFFER_MPI_WrRaw()</code>	Write an undecorated MPI value to the buffer.
<code>CRYPTO_BUFFER_MPI_WrRawLE()</code>	Write an undecorated MPI value to the buffer.
Mark-then-patch functions	
<code>CRYPTO_BUFFER_Mark()</code>	Mark position in buffer.
<code>CRYPTO_BUFFER_MarkU8()</code>	Mark position of U8 in buffer.
<code>CRYPTO_BUFFER_MarkU16()</code>	Mark position of U16 in buffer.
<code>CRYPTO_BUFFER_MarkU24()</code>	Mark position of U24 in buffer.
<code>CRYPTO_BUFFER_MarkU32()</code>	Mark position of U32 in buffer.
<code>CRYPTO_BUFFER_PatchU8()</code>	Patch marked U8.
<code>CRYPTO_BUFFER_PatchU16BE()</code>	Patch marked U16, big endian.
<code>CRYPTO_BUFFER_PatchU24BE()</code>	Patch marked U24, big endian.
<code>CRYPTO_BUFFER_PatchU32BE()</code>	Patch marked U32, big endian.
<code>CRYPTO_BUFFER_PatchU16LE()</code>	Patch marked U16, little endian.
<code>CRYPTO_BUFFER_PatchU24LE()</code>	Patch marked U24, little endian.
<code>CRYPTO_BUFFER_PatchU32LE()</code>	Patch marked U32, little endian.
Insertion functions	
<code>CRYPTO_BUFFER_Reserve()</code>	Reserve a number of bytes in the buffer.
<code>CRYPTO_BUFFER_Insert()</code>	Insert octet string.

20.1.1.1 CRYPTO_BUFFER_Copy()

Description

Copy object to buffer.

Prototype

```
void *CRYPTO_BUFFER_Copy(      CRYPTO_BUFFER * pSelf,
                               const void      * pData,
                               unsigned        DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pData	Pointer to octet string to copy to buffer.
DataLen	Octet length of the octet string.

Return value

NULL Pointer to the start of copied object (within buffer).
= NULL Buffer cannot accommodate the requested number of bytes.

Additional information

The returned pointer is not guaranteed to be aligned to the maximal addressable unit, it has byte granularity.

20.1.1.2 CRYPTO_BUFFER_Cursor()

Description

Return front position of buffer.

Prototype

```
U8 *CRYPTO_BUFFER_Cursor(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

Pointer to next byte to be written.

20.1.1.3 CRYPTO_BUFFER_CursorDistance()

Description

Return cursor distance beyond mark.

Prototype

```
unsigned CRYPTO_BUFFER_CursorDistance(CRYPTO_BUFFER * pSelf,  
                                      void           * pMark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pMark	A previous cursor position within the buffer.

Return value

Distance between the buffer's cursor position and the previous cursor position.

20.1.1.4 CRYPTO_BUFFER_CursorIndex()

Description

Return the cursor position.

Prototype

```
unsigned CRYPTO_BUFFER_CursorIndex(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer to query.

Return value

Position of the cursor within the buffer. This is usually the maximum number of bytes successfully written to the buffer.

20.1.1.5 CRYPTO_BUFFER_Data()

Description

Return pointer to start of buffer.

Prototype

```
U8 *CRYPTO_BUFFER_Data(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

Pointer to first byte of underlying buffer.

20.1.1.6 CRYPTO_BUFFER_Init()

Description

Initialize write buffer.

Prototype

```
void CRYPTO_BUFFER_Init(CRYPTO_BUFFER * pSelf,  
                        U8          * pOutput,  
                        unsigned      OutputLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer to initialize.
pOutput	Pointer to underlying octet accumulation buffer.
OutputLen	Octet length of the underlying octet accumulation buffer.

20.1.1.7 CRYPTO_BUFFER_Insert()

Description

Insert octet string.

Prototype

```
void CRYPTO_BUFFER_Insert(      CRYPTO_BUFFER * pSelf,
                                unsigned        Mark,
                                const void     * pData,
                                unsigned        DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position where the insertion takes place.
pData	Pointer to object to insert into buffer.
DataLen	Octet length of the object to insert.

20.1.1.8 CRYPTO_BUFFER_MPI_WrCounted()

Description

Write a length-counted MPI to the buffer.

Prototype

```
void CRYPTO_BUFFER_MPI_WrCounted(      CRYPTO_BUFFER * pSelf,  
                                      const CRYPTO_MPI    * pMPI);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pMPI	Pointer to MPU to write to buffer.

Additional information

The MPI is written with a 32-bit network-order length (L) followed by L bytes of the integer in network byte order. The integer is written such that the most significant byte never has its most significant bit set, i.e. it is always seen as positive.

This API is due for removal.

20.1.1.9 CRYPTO_BUFFER_MPI_WrRaw()

Description

Write an undecorated MPI value to the buffer.

Prototype

```
void CRYPTO_BUFFER_MPI_WrRaw(      CRYPTO_BUFFER * pSelf,
                                    const CRYPTO_MPI     * pMPI,
                                    unsigned             Len);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to buffer.
<code>pMPI</code>	Pointer to MPI to write to buffer.
<code>Len</code>	Number of bytes that the MPI takes.

Additional information

The MPI is written in a field of `Len` bytes in network byte order.

20.1.1.10 CRYPTO_BUFFER_MPI_WrRawLE()

Description

Write an undecorated MPI value to the buffer.

Prototype

```
void CRYPTO_BUFFER_MPI_WrRawLE( CRYPTO_BUFFER * pSelf,
                                const CRYPTO_MPI      * pMPI,
                                unsigned                Len);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to buffer.
<code>pMPI</code>	Pointer to MPI to write to buffer.
<code>Len</code>	Number of bytes that the MPI takes.

Additional information

The MPI is written in a field of `Len` bytes in PC byte order.

20.1.1.11 CRYPTO_BUFFER_Mark()

Description

Mark position in buffer.

Prototype

```
unsigned CRYPTO_BUFFER_Mark(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

A mark indicating the current cursor position within the buffer.

20.1.1.12 CRYPTO_BUFFER_MarkU8()

Description

Mark position of U8 in buffer.

Prototype

```
unsigned CRYPTO_BUFFER_MarkU8(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

A mark indicating the position of the U8 in the buffer which can be patched using CRYPTO_BUFFER_PatchU8()

20.1.1.13 CRYPTO_BUFFER_MarkU16()

Description

Mark position of U16 in buffer.

Prototype

```
unsigned CRYPTO_BUFFER_MarkU16(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

A mark indicating the position of the U16 in the buffer which can be patched using CRYPTO_BUFFER_PatchU16BE() or CRYPTO_BUFFER_PatchU16LE().

20.1.1.14 CRYPTO_BUFFER_MarkU24()

Description

Mark position of U24 in buffer.

Prototype

```
unsigned CRYPTO_BUFFER_MarkU24(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

A mark indicating the position of the U24 in the buffer which can be patched using CRYPTO_BUFFER_PatchU24BE() or CRYPTO_BUFFER_PatchU24LE().

20.1.1.15 CRYPTO_BUFFER_MarkU32()

Description

Mark position of U32 in buffer.

Prototype

```
unsigned CRYPTO_BUFFER_MarkU32(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

Return value

A mark indicating the position of the U32 in the buffer which can be patched using CRYPTO_BUFFER_PatchU32BE() or CRYPTO_BUFFER_PatchU32LE().

20.1.1.16 CRYPTO_BUFFER_Overflow()

Description

Inquire the number of bytes that were not written.

Prototype

```
unsigned CRYPTO_BUFFER_Overflow(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer to query.

Return value

- = 0 Buffer has not overflowed.
- > 0 Buffer has overflowed, number of bytes dropped from buffer.

20.1.1.17 CRYPTO_BUFFER_PatchU8()

Description

Patch marked U8.

Prototype

```
U8 CRYPTO_BUFFER_PatchU8(CRYPTO_BUFFER * pSelf,  
                           unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U8 within buffer.

Return value

Value patched into buffer.

Additional information

The U8 at the marked position is updated with a U8. The value patched into the buffer is the distance between the end of the U8 and the cursor, i.e. the length of the data following the U8 up to the cursor.

20.1.1.18 CRYPTO_BUFFER_PatchU16BE()

Description

Patch marked U16, big endian.

Prototype

```
U16 CRYPTO_BUFFER_PatchU16BE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U16 within buffer.

Return value

Value patched into buffer.

Additional information

The U16 at the marked position is updated with a U16 in network byte order. The value patched into the buffer is the distance between the end of the U16 and the cursor, i.e. the length of the data following the U16 up to the cursor.

20.1.1.19 CRYPTO_BUFFER_PatchU24BE()

Description

Patch marked U24, big endian.

Prototype

```
U32 CRYPTO_BUFFER_PatchU24BE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U24 within buffer.

Return value

Value patched into buffer.

Additional information

The U24 at the marked position is updated with a U24 in network byte order. The value patched into the buffer is the distance between the end of the U24 and the cursor, i.e. the length of the data following the U24 up to the cursor.

20.1.1.20 CRYPTO_BUFFER_PatchU32BE()

Description

Patch marked U32, big endian.

Prototype

```
U32 CRYPTO_BUFFER_PatchU32BE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U32 within buffer.

Return value

Value patched into buffer.

Additional information

The U32 at the marked position is updated with a U32 in network byte order. The value patched into the buffer is the distance between the end of the U32 and the cursor, i.e. the length of the data following the U32 up to the cursor.

20.1.1.21 CRYPTO_BUFFER_PatchU16LE()

Description

Patch marked U16, little endian.

Prototype

```
U16 CRYPTO_BUFFER_PatchU16LE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U16 within buffer.

Return value

Value patched into buffer.

Additional information

The U16 at the marked position is updated with a U16 in PC byte order. The value patched into the buffer is the distance between the end of the U16 and the cursor, i.e. the length of the data following the U16 up to the cursor.

20.1.1.22 CRYPTO_BUFFER_PatchU24LE()

Description

Patch marked U24, little endian.

Prototype

```
U32 CRYPTO_BUFFER_PatchU24LE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U24 within buffer.

Return value

Value patched into buffer.

Additional information

The U24 at the marked position is updated with a U24 in PC byte order. The value patched into the buffer is the distance between the end of the U24 and the cursor, i.e. the length of the data following the U24 up to the cursor.

20.1.1.23 CRYPTO_BUFFER_PatchU32LE()

Description

Patch marked U32, little endian.

Prototype

```
U32 CRYPTO_BUFFER_PatchU32LE(CRYPTO_BUFFER * pSelf,  
                               unsigned           Mark);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Mark	Marked position of U32 within buffer.

Return value

Value patched into buffer.

Additional information

The U32 at the marked position is updated with a U32 in PC byte order. The value patched into the buffer is the distance between the end of the U32 and the cursor, i.e. the length of the data following the U32 up to the cursor.

20.1.1.24 CRYPTO_BUFFER_Reserve()

Description

Reserve a number of bytes in the buffer.

Prototype

```
U8 *CRYPTO_BUFFER_Reserve(CRYPTO_BUFFER * pSelf,  
                           unsigned           Len);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Len	Number of octets to reserve.

Return value

NULL Pointer to the start of the reserved memory.
= NULL Buffer cannot accommodate the requested number of bytes.

Additional information

The returned pointer is not guaranteed to be aligned to the maximal addressable unit, it has byte granularity.

20.1.1.25 CRYPTO_BUFFER_Reset()

Description

Reset buffer to empty state.

Prototype

```
void CRYPTO_BUFFER_Reset(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.

20.1.1.26 CRYPTO_BUFFER_SetCursor()

Description

Set the cursor index.

Prototype

```
void CRYPTO_BUFFER_SetCursor(CRYPTO_BUFFER * pSelf,  
                           void           * pCursor);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pCursor	New cursor position.

Additional information

The cursor is constrained to remain within the buffer.

20.1.1.27 CRYPTO_BUFFER_SetCursorPosition()

Description

Set the cursor index.

Prototype

```
void CRYPTO_BUFFER_SetCursorPosition(CRYPTO_BUFFER * pSelf,  
                                    unsigned           Index);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Index	New index position.

Additional information

The cursor is constrained to remain within the buffer.

20.1.1.28 CRYPTO_BUFFER_Skip()

Description

Advance write pointer in buffer.

Prototype

```
void CRYPTO_BUFFER_Skip(CRYPTO_BUFFER * pSelf,  
                        unsigned          Count);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Count	Number of octets to skip.

20.1.1.29 CRYPTO_BUFFER_SpaceLeft()

Description

Inquire number of bytes that can be written.

Prototype

```
unsigned CRYPTO_BUFFER_SpaceLeft(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer to query.

Return value

Maximum number of bytes that can be written without overflow.

20.1.1.30 CRYPTO_BUFFER_Status()

Description

Return status indicating buffer overflow.

Prototype

```
int CRYPTO_BUFFER_Status(CRYPTO_BUFFER * pSelf);
```

Parameters

Parameter	Description
<code>pSelf</code>	Pointer to buffer to query.

Return value

- < 0 Buffer has overflowed.
- = 0 Buffer has not overflowed.

20.1.1.31 CRYPTO_BUFFER_Wr()

Description

Write object to buffer.

Prototype

```
void CRYPTO_BUFFER_Wr(      CRYPTO_BUFFER * pSelf,
                           const void          * pData,
                           unsigned             DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pData	Pointer to object to write to buffer.
DataLen	Octet length of the object.

20.1.1.32 CRYPTO_BUFFER_WrCntU32BE()

Description

Write counted string.

Prototype

```
void CRYPTO_BUFFER_WrCntU32BE( CRYPTO_BUFFER * pSelf,
                                const void      * pData,
                                U32             DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
pData	Pointer to object to write.
DataLen	Octet length of the object to write.

Additional information

The object is written to the buffer and is preceded by a 32-bit length in network byte order.

20.1.1.33 CRYPTO_BUFFER_WrLogical()

Description

Write a block the write buffer and apply logic operation.

Prototype

```
void CRYPTO_BUFFER_WrLogical( CRYPTO_BUFFER * pSelf,
                               CRYPTO_LOGIC_OP Op,
                               const U8          * pData,
                               unsigned         DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Op	Logic operation to apply.
pData	Pointer to object to write to buffer.
DataLen	Octet length of the object.

20.1.1.34 CRYPTO_BUFFER_WrMultiU8()

Description

Write multiple identical bytes.

Prototype

```
void CRYPTO_BUFFER_WrMultiU8(CRYPTO_BUFFER * pSelf,  
                           U8          Data,  
                           unsigned    Count);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Data	Value to write.
Count	Number of times to write Data to buffer.

20.1.1.35 CRYPTO_BUFFER_WrStr()

Description

Write string to buffer.

Prototype

```
void CRYPTO_BUFFER_WrStr(      CRYPTO_BUFFER * pSelf,  
                           const char          * sText);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
sText	Pointer to string.

20.1.1.36 CRYPTO_BUFFER_WrStrLn()

Description

Write string and end-of-line to buffer.

Prototype

```
void CRYPTO_BUFFER_WrStrLn(      CRYPTO_BUFFER * pSelf,
                                const char          * sText);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
sText	Pointer to string.

20.1.1.37 CRYPTO_BUFFER_WrU16BE()

Description

Write a 16-bit unsigned integer in network byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU16BE(CRYPTO_BUFFER * pSelf,  
                           U16           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.38 CRYPTO_BUFFER_WrU16LE()

Description

Write a 16-bit unsigned integer in PC byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU16LE(CRYPTO_BUFFER * pSelf,  
                           U16           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.39 CRYPTO_BUFFER_WrU24BE()

Description

Write a 24-bit unsigned integer in network byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU24BE(CRYPTO_BUFFER * pSelf,  
                           U32           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.40 CRYPTO_BUFFER_WrU24LE()

Description

Write a 24-bit unsigned integer in PC byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU24LE(CRYPTO_BUFFER * pSelf,  
                           U32           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.41 CRYPTO_BUFFER_WrU32BE()

Description

Write a 32-bit unsigned integer in network byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU32BE(CRYPTO_BUFFER * pSelf,  
                           U32           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.42 CRYPTO_BUFFER_WrU32LE()

Description

Write a 32-bit unsigned integer in PC byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU32LE(CRYPTO_BUFFER * pSelf,  
                           U32           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.43 CRYPTO_BUFFER_WrU64BE()

Description

Write a 64-bit unsigned integer in network byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU64BE(CRYPTO_BUFFER * pSelf,  
                           U64           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.44 CRYPTO_BUFFER_WrU64LE()

Description

Write a 64-bit unsigned integer in PC byte order to the write buffer.

Prototype

```
void CRYPTO_BUFFER_WrU64LE(CRYPTO_BUFFER * pSelf,  
                           U64           Data);
```

Parameters

Parameter	Description
pSelf	Buffer to write into.
Data	Value to write.

20.1.1.45 CRYPTO_BUFFER_WrU8()

Description

Write a single byte to the buffer.

Prototype

```
void CRYPTO_BUFFER_WrU8(CRYPTO_BUFFER * pSelf,  
                         U8             Data);
```

Parameters

Parameter	Description
pSelf	Pointer to buffer.
Data	Value to write.

20.2 TLV parsing

20.2.1 Type-safe API

The following table lists the TLV API functions.

Function	Description
Management	
<code>CRYPTO_TLV_Init()</code>	Initialize TLV.
<code>CRYPTO_TLV_Prepares()</code>	Prepare TLV for parsing.
<code>CRYPTO_TLV_Copy()</code>	Copy TLV.
<code>CRYPTO_TLV_Trim()</code>	Reduce the length of a TLV.
<code>CRYPTO_TLV_Close()</code>	Close a TLV and prohibit reading from it.
<code>CRYPTO_TLV_ForceClose()</code>	Force closure of a TLV discarding unread data.
Look-ahead	
<code>CRYPTO_TLV_PeekTag()</code>	Return tag at the cursor position.
<code>CRYPTO_TLV_PeekU8()</code>	Read an octet from TLV but do not advance.
Parsing	
<code>CRYPTO_TLV_Read()</code>	Parse an octet string from a TLV.
<code>CRYPTO_TLV_ReadU8()</code>	Parse the next octet from a TLV.
<code>CRYPTO_TLV_ReadU16()</code>	Read a 16-bit integer from TLV in network byte order.
<code>CRYPTO_TLV_ReadU24()</code>	Read a 24-bit integer from TLV in network byte order.
<code>CRYPTO_TLV_ReadU32()</code>	Read a 32-bit integer from TLV in network byte order.
<code>CRYPTO_TLV_ParseTagAndLength()</code>	Parse the tag and length part of a TLV.
<code>CRYPTO_TLV_ParseINTEGER()</code>	Parse a multiprecision integer encoded as an ASN.1 INTEGER.
<code>CRYPTO_TLV_SkipBytes()</code>	Parse, but do not store, an octet string from a TLV.
<code>CRYPTO_TLV_SkipNL()</code>	Skip all up to and including newline.
<code>CRYPTO_TLV_SkipINTEGER()</code>	Skip over an integer.
<code>CRYPTO_TLV_EnsureBytes()</code>	Query remaining length of TLV.
<code>CRYPTO_TLV_MPI_LoadBytes()</code>	Read a multi-byte unsigned integer in network byte order.
<code>CRYPTO_TLV_MPI_LoadBytesLE()</code>	Read a multi-byte unsigned integer in PC byte order.
Conditional parsing	
<code>CRYPTO_TLV_Accept()</code>	Conditionally accept data from TLV.
<code>CRYPTO_TLV_AcceptStr()</code>	Conditionally accept string from TLV.
<code>CRYPTO_TLV_Capture()</code>	Capture the value part of a TLV.
<code>CRYPTO_TLV_CaptureValue()</code>	Capture an octet string from a TLV into a TLV.
<code>CRYPTO_TLV_CaptureTo()</code>	Read from a TLV up to a matching octet delimiter or end of data.

Function	Description
<code>CRYPTO_TLV_CaptureToNL()</code>	Read from a TLV up to the end of line.
Queries	
<code>CRYPTO_TLV_CheckNull()</code>	Parse an ASN.1 NULL TLV.
<code>CRYPTO_TLV_IsCompletelyRead()</code>	Query if TLV has no data remaining to be read.
<code>CRYPTO_TLV_IsValueEqual()</code>	Match a TLV value.
<code>CRYPTO_TLV_GetNumUnread()</code>	Query number of unread octets in TLV.
<code>CRYPTO_TLV_MatchValues()</code>	Are two TLV values identical?

20.2.1.1 CRYPTO_TLV_Accept()

Description

Conditionally accept data from TLV.

Prototype

```
int CRYPTO_TLV_Accept(      CRYPTO_TLV * pSelf,
                           const void      * pData,
                           unsigned        DataLen);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pData	Pointer to octet string to accept.
DataLen	Octet length of the octet string.

Return value

= 0 No match and not accepted.
≠ 0 Match and accepted.

20.2.1.2 CRYPTO_TLV_AcceptStr()

Description

Conditionally accept string from TLV.

Prototype

```
int CRYPTO_TLV_AcceptStr(      CRYPTO_TLV * pSelf,  
                           const char      * sText);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
sText	String to accept.

Return value

- = 0 No match and not accepted.
- ≠ 0 Match and accepted.

20.2.1.3 CRYPTO_TLV_Capture()

Description

Capture the value part of a TLV.

Prototype

```
int CRYPTO_TLV_Capture(CRYPTO_TLV * pSelf,  
                      CRYPTO_TLV * pValue,  
                      unsigned      Tag);
```

Parameters

Parameter	Description
pSelf	TLV to capture TLV from.
pValue	Pointer to TLV that receives the captured value component of the TLV pSelf.
Tag	Tag to match.

Return value

≥ 0 Success, value part of TLV captured.
< 0 Error, TLV is too short.

20.2.1.4 CRYPTO_TLV_CaptureTo()

Description

Read from a TLV up to a matching octet delimiter or end of data.

Prototype

```
void CRYPTO_TLV_CaptureTo(CRYPTO_TLV * pSelf,  
                           CRYPTO_TLV * pString,  
                           U8           Delim);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pString	TLV that will receive the data.
Delim	The delimiter that terminates the octet string.

20.2.1.5 CRYPTO_TLV_CaptureToNL()

Description

Read from a TLV up to the end of line.

Prototype

```
void CRYPTO_TLV_CaptureToNL(CRYPTO_TLV * pSelf,  
                           CRYPTO_TLV * pString);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pString	TLV that will receive the data.

20.2.1.6 CRYPTO_TLV_CaptureValue()

Description

Capture an octet string from a TLV into a TLV.

Prototype

```
int CRYPTO_TLV_CaptureValue(CRYPTO_TLV * pSelf,  
                           CRYPTO_TLV * pValue,  
                           U32           Len);
```

Parameters

Parameter	Description
pSelf	TLV to capture octet string from.
pValue	Pointer to TLV that receives the captured octet string.
Len	Length of the octet string to capture from pSelf.

Return value

≥ 0 Success, octet string captured.
< 0 Error, TLV is too short.

Additional information

The value part of pSelf is not copied into pValue, but rather pValue points to the appropriate substring within the TLV pSelf.

20.2.1.7 CRYPTO_TLV_CheckNull()

Description

Parse an ASN.1 NULL TLV.

Prototype

```
int CRYPTO_TLV_CheckNull(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	Container TLV.

Return value

- ≥ 0 NULL tag correctly read with null value.
- < 0 Error reading TLV sequence or incorrect tag.

20.2.1.8 CRYPTO_TLV_Close()

Description

Close a TLV and prohibit reading from it.

Prototype

```
int CRYPTO_TLV_Close(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to close.

Return value

≥ 0 Success, TLV is successfully closed.

< 0 Error, TLV has data remaining to be read and is considered malformed.

Additional information

In order to prevent attacks on software using TLVs, clients should call `CRYPTO_TLV_Close` when they are finished reading a TLV to ensure that all data has been correctly read from the subject TLV.

20.2.1.9 CRYPTO_TLV_Copy()

Description

Copy TLV.

Prototype

```
void CRYPTO_TLV_Copy(      CRYPTO_TLV * pSelf,  
                           const CRYPTO_TLV * pItem);
```

Parameters

Parameter	Description
pSelf	Pointer to TLV that receives the copy.
pItem	Pointer to TLV that is copied.

20.2.1.10 CRYPTO_TLV_EnsureBytes()

Description

Query remaining length of TLV.

Prototype

```
int CRYPTO_TLV_EnsureBytes(CRYPTO_TLV * pSelf,  
                           unsigned     Len);
```

Parameters

Parameter	Description
pSelf	TLV to query.
Len	Number of bytes requires.

Return value

- ≥ 0 Success, at least Length bytes remain to be parsed.
- < 0 Error, fewer than Length bytes remain to be parsed.

20.2.1.11 CRYPTO_TLV_ForceClose()

Description

Force closure of a TLV discarding unread data.

Prototype

```
void CRYPTO_TLV_ForceClose(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to close.

20.2.1.12 CRYPTO_TLV_GetNumUnread()

Description

Query number of unread octets in TLV.

Prototype

```
unsigned CRYPTO_TLV_GetNumUnread(const CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to query.

Return value

Number of unread octets remaining in TLV.

20.2.1.13 CRYPTO_TLV_Init()

Description

Initialize TLV.

Prototype

```
void CRYPTO_TLV_Init(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to TLV to initialize.

Additional information

The TLV is initialized with length zero and a zero tag. The Value data pointer is set to null.

20.2.1.14 CRYPTO_TLV_IsCompletelyRead()

Description

Query if TLV has no data remaining to be read.

Prototype

```
int CRYPTO_TLV_IsCompletelyRead(const CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to query.

Return value

- ≠ 0 TLV has been completely read, no data remaining.
- = 0 Data remains to be read from TLV.

20.2.1.15 CRYPTO_TLV_IsValueEqual()

Description

Match a TLV value.

Prototype

```
int CRYPTO_TLV_IsValueEqual( CRYPTO_TLV * pSelf,
                             const U8      * pValue,
                             unsigned     ValueLen);
```

Parameters

Parameter	Description
pSelf	Pointer to TLV.
pValue	Pointer to value to match.
ValueLen	Octet length of value.

Return value

- # 0 TLV values are identical.
- = 0 TLV values are nonidentical.

20.2.1.16 CRYPTO_TLV_MPI_LoadBytes()

Description

Read a multi-byte unsigned integer in network byte order.

Prototype

```
int CRYPTO_TLV_MPI_LoadBytes(CRYPTO_TLV * pSelf,  
                             CRYPTO_MPI * pMPI,  
                             unsigned     Len);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pMPI	Pointer to an initialized MPI that will receive the data.
Len	Octet length of the MPI to load as unsigned.

Return value

≥ 0 Value correctly read.
< 0 Processing error.

Additional information

The data is read from the TLV in network byte order.

20.2.1.17 CRYPTO_TLV_MPI_LoadBytesLE()

Description

Read a multi-byte unsigned integer in PC byte order.

Prototype

```
int CRYPTO_TLV_MPI_LoadBytesLE(CRYPTO_TLV * pSelf,  
                                CRYPTO_MPI * pMPI,  
                                unsigned     Len);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pMPI	Pointer to an initialized MPI that will receive the data.
Len	Octet length of the MPI to load as unsigned.

Return value

≥ 0 Value correctly read.
< 0 Processing error.

Additional information

The data is read from the TLV in little-endian byte order.

20.2.1.18 CRYPTO_TLV_MatchValues()

Description

Are two TLV values identical?

Prototype

```
int CRYPTO_TLV_MatchValues(const CRYPTO_TLV * pTLV0,  
                           const CRYPTO_TLV * pTLV1);
```

Parameters

Parameter	Description
pTLV0	TLV #0.
pTLV1	TLV #1.

Return value

- = 0 - TLVs have identical value fields.
- ≠ 0 - TLVs have differing value fields.

20.2.1.19 CRYPTO_TLV_ParseINTEGER()

Description

Parse a multiprecision integer encoded as an ASN.1 INTEGER.

Prototype

```
int CRYPTO_TLV_ParseINTEGER(CRYPTO_TLV * pSelf,  
                           CRYPTO_MPI * pMPI);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pMPI	Pointer to an initialized MPI that will receive the data.

Return value

- ≥ 0 Value correctly read.
- < 0 Error reading value (tag mismatch or malformed ASN.1).

Additional information

The data is read from the TLV in network byte order. The ASN.1 tag must specify an INTEGER data type.

20.2.1.20 CRYPTO_TLV_ParseTagAndLength()

Description

Parse the tag and length part of a TLV.

Prototype

```
int CRYPTO_TLV_ParseTagAndLength(CRYPTO_TLV * pSelf,  
                                  unsigned ExpectedTag,  
                                  unsigned * pLength);
```

Parameters

Parameter	Description
pSelf	TLV to parse.
ExpectedTag	Expected tag.
pLength	Pointer to object that receives the parsed length.

Return value

- ≥ 0 Success.
- < 0 Error reading TLV sequence or incorrect tag.

Additional information

This function leaves the parsing state ready to parse the value field with subsequent function calls.

20.2.1.21 CRYPTO_TLV_PeekTag()

Description

Return tag at the cursor position.

Prototype

```
unsigned CRYPTO_TLV_PeekTag(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV.

Return value

TLV tag as an unsigned integer.

Additional information

Tags can be more than four bytes in length, and this code does not cater for such tags.

20.2.1.22 CRYPTO_TLV_PeekU8()

Description

Read an octet from TLV but do not advance.

Prototype

```
int CRYPTO_TLV_PeekU8(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to read from.

Return value

≥ 0 Success, 8-bit value read from TLV.
< 0 Error, TLV is too short.

20.2.1.23 CRYPTO_TLV_Prepares()

Description

Prepare TLV for parsing.

Prototype

```
void CRYPTO_TLV_Prepares(      CRYPTO_TLV * pSelf,
                               const void      * pData,
                               unsigned        NumBytesData);
```

Parameters

Parameter	Description
pSelf	Pointer to TLV to prepare.
pData	Pointer to octet string covering the TLV.
NumBytesData	Octet length of the string to parse.

20.2.1.24 CRYPTO_TLV_Read()

Description

Parse an octet string from a TLV.

Prototype

```
int CRYPTO_TLV_Read(CRYPTO_TLV * pSelf,  
                     void      * pDest,  
                     unsigned   Length);
```

Parameters

Parameter	Description
pSelf	TLV to parse.
pDest	Pointer to object that will receive the data.
Length	Length of octet string to parse from the TLV.

Return value

- ≥ 0 Success, octet stream read correctly.
- < 0 Error, read beyond end of TLV.

Additional information

This function leaves the parsing state ready to parse the value field with subsequent function calls.

20.2.1.25 CRYPTO_TLV_ReadU16()

Description

Read a 16-bit integer from TLV in network byte order.

Prototype

```
I32 CRYPTO_TLV_ReadU16(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to read from.

Return value

≥ 0 Success, 16-bit value read from TLV.
 < 0 Error, TLV is too short.

20.2.1.26 CRYPTO_TLV_ReadU24()

Description

Read a 24-bit integer from TLV in network byte order.

Prototype

```
int CRYPTO_TLV_ReadU24(CRYPTO_TLV * pSelf,  
                        U32          * pData);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pData	Pointer to object that will receive the data.

Return value

≥ 0 Success, 24-bit value read from TLV.
< 0 Error, TLV is too short.

20.2.1.27 CRYPTO_TLV_ReadU32()

Description

Read a 32-bit integer from TLV in network byte order.

Prototype

```
int CRYPTO_TLV_ReadU32(CRYPTO_TLV * pSelf,  
                        U32          * pData);
```

Parameters

Parameter	Description
pSelf	TLV to read from.
pData	Pointer to object that will receive the data.

Return value

- ≥ 0 Success, 32-bit value read from TLV.
- < 0 Error, TLV is too short.

20.2.1.28 CRYPTO_TLV_ReadU8()

Description

Parse the next octet from a TLV.

Prototype

```
int CRYPTO_TLV_ReadU8(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to parse.

Return value

≥ 0 Success, octet read from TLV stream.
< 0 Error, read beyond end of TLV.

Additional information

This function leaves the parsing state ready to parse the value field with subsequent function calls.

20.2.1.29 CRYPTO_TLV_SkipBytes()

Description

Parse, but do not store, an octet string from a TLV.

Prototype

```
int CRYPTO_TLV_SkipBytes(CRYPTO_TLV * pSelf,  
                         unsigned     Len);
```

Parameters

Parameter	Description
pSelf	TLV to parse.
Len	Length of octet string to skip.

Return value

- ≥ 0 Success, octet stream skipped correctly.
- < 0 Error, read beyond end of TLV.

20.2.1.30 CRYPTO_TLV_SkipINTEGER()

Description

Skip over an integer.

Prototype

```
int CRYPTO_TLV_SkipINTEGER(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to read from.

Return value

- ≥ 0 Value correctly read.
- < 0 Error skipping integer (tag mismatch or malformed ASN.1).

Additional information

The data is read from the TLV in network byte order. The ASN.1 tag must specify an INTEGER data type.

20.2.1.31 CRYPTO_TLV_SkipNL()

Description

Skip all up to and including newline.

Prototype

```
void CRYPTO_TLV_SkipNL(CRYPTO_TLV * pSelf);
```

Parameters

Parameter	Description
pSelf	TLV to read from.

20.2.1.32 CRYPTO_TLV_Trim()

Description

Reduce the length of a TLV.

Prototype

```
int CRYPTO_TLV_Trim(CRYPTO_TLV * pSelf,  
                      unsigned     Len);
```

Parameters

Parameter	Description
pSelf	TLV to trim.
Len	Require new octet length of TLV.

Return value

- ≥ 0 Success, octet stream trimmed.
- < 0 Error, required length exceeds TLV length.

20.3 Counters

20.3.1 Type-safe API

The following table lists the counter API functions.

Function	Description
Management functions	
CRYPTO_IncCTRBE()	Increment counter, network byte order.
CRYPTO_IncCTRBE_TrapOverflow()	Increment counter, network byte order, trap on overflow.
CRYPTO_IncCTRLE()	Increment a counter, little-endian byte order.
CRYPTO_IncCTRLE_TrapOverflow()	Increment counter, PC byte order, trap on overflow.

20.3.1.1 CRYPTO_IncCTRBE()

Description

Increment counter, network byte order.

Prototype

```
unsigned CRYPTO_IncCTRBE(U8      * pCTR,  
                          unsigned CTRLen,  
                          unsigned N);
```

Parameters

Parameter	Description
pCTR	Pointer to the MSB of the counter.
CTRLen	Octet length of the counter.
N	Increment to the counter, ≤ 255 .

Return value

Carry out from counter.

20.3.1.2 CRYPTO_IncCTRBE_TrapOverflow()

Description

Increment counter, network byte order, trap on overflow.

Prototype

```
int CRYPTO_IncCTRBE_TrapOverflow(U8      * pCTR,  
                                  unsigned CTRLen,  
                                  unsigned N);
```

Parameters

Parameter	Description
pCTR	Pointer to the MSB of the counter.
CTRLen	Octet length of the counter.
N	Increment to the counter, ≤ 255 .

Return value

≥ 0 No overflow.
 < 0 Overflow.

20.3.1.3 CRYPTO_IncCTRLE()

Description

Increment a counter, little-endian byte order.

Prototype

```
unsigned CRYPTO_IncCTRLE(U8      * pCTR,  
                          unsigned CTRLen,  
                          unsigned N);
```

Parameters

Parameter	Description
pCTR	Pointer to the LSB of the counter.
CTRLen	Counter length in bytes.
N	Increment to the counter, ≤ 255 .

Return value

Carry out from counter.

20.3.1.4 CRYPTO_IncCTRLE_TrapOverflow()

Description

Increment counter, PC byte order, trap on overflow.

Prototype

```
int CRYPTO_IncCTRLE_TrapOverflow(U8      * pCTR,  
                                  unsigned CTRLen,  
                                  unsigned N);
```

Parameters

Parameter	Description
pCTR	Pointer to the LSB of the counter.
CTRLen	Octet length of the counter.
N	Increment to the counter, ≤ 255 .

Return value

≥ 0 No overflow.
 < 0 Overflow.

Chapter 21

Benchmarks

This section describes the various benchmarking applications that are distributed with emCrypt. All timings are made with an STM32F756-EVAL board running at 200 MHz using SEGGER Embedded Studio and the clang compiler (unless otherwise noted).

21.1 Ciphers

21.1.1 CRYPTO_Bench_AES.c

This application benchmarks the configured performance of AES. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.1.1.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
AES Benchmark compiled Mar 19 2018 16:29:26

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION           = 22400 [2.24]
Config: CRYPTO_CONFIG_AES_OPTIMIZE = 7
Config: CRYPTO_CONFIG_AES_HW_OPTIMIZE = 1
Config: CRYPTO_CONFIG_GCM_OPTIMIZE = 0

+-----+-----+-----+-----+
|     | ECB   MB/s | CBC   MB/s |
| Cipher | Bits | Enc Dec | Enc Dec |
+-----+-----+-----+
| AES    | 128  | 3.72  | 3.48  | 2.85  | 2.68  |
| AES (HW)| 128  | 46.49 | 46.47 | 46.51 | 43.57 |
| AES    | 192  | 2.80  | 2.67  | 2.46  | 2.32  |
| AES (HW)| 192  | 42.57 | 42.58 | 42.61 | 40.19 |
| AES    | 256  | 2.43  | 2.32  | 2.16  | 2.05  |
| AES (HW)| 256  | 42.56 | 42.57 | 42.61 | 40.08 |
+-----+-----+-----+
|     | GCM   MB/s | CCM   MB/s |
| Cipher | Bits | Enc Dec | Enc Dec |
+-----+-----+-----+
| AES    | 128  | 0.11  | 0.11  | 1.38  | 1.39  |
| AES (HW)| 128  | 0.11  | 0.11  | 3.92  | 3.92  |
| AES    | 192  | 0.11  | 0.11  | 1.20  | 1.20  |
| AES (HW)| 192  | 0.11  | 0.11  | 3.86  | 3.85  |
| AES    | 256  | 0.12  | 0.12  | 1.05  | 1.06  |
| AES (HW)| 256  | 0.12  | 0.12  | 3.86  | 3.85  |
+-----+-----+-----+

Benchmark complete
```

21.1.1.2 Complete listing

```
*****
*(c) SEGGER Microcontroller GmbH *
* The Embedded Experts *
* www.segger.com *
*****  

----- END-OF-HEADER -----  

File       : CRYPTO_Bench_AES.c
Purpose    : Benchmark AES implementation.  

*/  

*****  

* #include section  

*  

*****  

*/  

#include "CRYPTO.h"
```

```
#include "SEGGER_SYS.h"

*****
*
*      Static data
*
*****
*/
static U8 _aTestMessage[1024] = { 0 };
static U8 _aAAD[13] = { 0 };

*****
*
*      Static code
*
*****
*/
/*
*      _ConvertTicksToSeconds()
*
*      Function description
*          Convert ticks to seconds.
*
*      Parameters
*          Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
*      Return value
*          Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

/*
*      _CipherBenchmark_ECB_CBC()
*
*      Function description
*          Benchmarks a cipher implementation.
*
*      Parameters
*          sAlgorithm - Cipher algorithm name.
*          pAPI - Pointer to cipher API.
*          KeySize - Cipher key size in bytes.
*/
static void _CipherBenchmark_ECB_CBC(const char *sAlgorithm, const CRYPTO_CIPHER_API *pAPI, unsigned KeySize)
{
    CRYPTO_AES_CONTEXT Context;
    U64 T0;
    U64 OneSecond;
    U8 aIV [16];
    U8 aKey[32];
    unsigned n;
    //
    CRYPTO_MEMZAP(aIV, sizeof(aIV));
    CRYPTO_MEMZAP(aKey, sizeof(aKey));
    //
    SEGGER_SYS_IO_Printf(" | %-12s | %4d | ", sAlgorithm, KeySize*8);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    // ECB encrypt
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInitEncrypt(&Context, aKey, KeySize);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        CRYPTO_CIPHER_ECB_Encrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), pAPI);
        n += sizeof(_aTestMessage);
    }
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
    //
    // ECB decrypt
}
```

```

//  

T0 = SEGGER_SYS_OS_GetTimer();  

n = 0;  

pAPI->pfInitDecrypt(&Context, aKey, KeySize);  

while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {  

  

    CRYPTO_CIPHER_ECB_Decrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), pAPI);  

    n += sizeof(_aTestMessage);  

}  

T0 = SEGGER_SYS_OS_GetTimer() - T0;  

SEGGER_SYS_IO_Printf("%7.2f |  

", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));  

//  

// CBC encrypt  

//  

T0 = SEGGER_SYS_OS_GetTimer();  

n = 0;  

pAPI->pfInitEncrypt(&Context, aKey, KeySize);  

while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {  

  

    CRYPTO_CIPHER_CBC_Encrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI);  

    n += sizeof(_aTestMessage);  

}  

T0 = SEGGER_SYS_OS_GetTimer() - T0;  

SEGGER_SYS_IO_Printf("%7.2f |  

", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));  

//  

// CBC decrypt  

//  

T0 = SEGGER_SYS_OS_GetTimer();  

n = 0;  

pAPI->pfInitDecrypt(&Context, aKey, KeySize);  

while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {  

  

    CRYPTO_CIPHER_CBC_Decrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI);  

    n += sizeof(_aTestMessage);  

}  

T0 = SEGGER_SYS_OS_GetTimer() - T0;  

SEGGER_SYS_IO_Printf("%7.2f |  

\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));  

}  

*****  

*  

*      _CipherBenchmark_GCM_CCM()  

*  

*  Function description  

*      Benchmarks a cipher implementation.  

*  

*  Parameters  

*      sAlgorithm - Cipher algorithm name.  

*      pAPI       - Pointer to cipher API.  

*      KeySize    - Cipher key size in bytes.  

*/  

static void _CipherBenchmark_GCM_CCM(const char *sAlgorithm, const CRYPTO_CIPHER_API *pAPI, unsigned KeySize)  

CRYPTO_AES_CONTEXT Context;  

U64          T0;  

U64          OneSecond;  

U8           aIV[16];  

U8           aKey[32];  

U8           aTag[16];  

unsigned      n;  

//  

CRYPTO_MEMZAP(aIV, sizeof(aIV));  

CRYPTO_MEMZAP(aKey, sizeof(aKey));  

//  

SEGGER_SYS_IO_Printf(" | %-12s | %4d | ", sAlgorithm, KeySize*8);  

OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);  

//  

// GCM encrypt  

//  

T0 = SEGGER_SYS_OS_GetTimer();  

n = 0;  

pAPI->pfInitEncrypt(&Context, aKey, KeySize);  

while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {  

  

    CRYPTO_AES_GCM_Encrypt(&Context, &_aTestMessage[0], &aTag[0], sizeof(aTag), &_aTestMessage[0], sizeof(_aTe

```

```

        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// GCM decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_AES_GCM_Decrypt(&Context, &aTestMessage[0], &aTag[0], sizeof(aTag), &aTestMessage[0], sizeof(_aTe
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f |
", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CCM encrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_AES_CCM_Encrypt(&Context, &aTestMessage[0], &aTag[0], 16, &aTestMessage[0], sizeof(_aTestMessage)
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CCM decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_AES_CCM_Decrypt(&Context, &aTestMessage[0], &aTag[0], 16, &aTestMessage[0], sizeof(_aTestMessage)
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f |
\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
*     _GetHWAPI()
*
* Function description
*     Returns hardware acceleration API for given key size.
*
* Parameters
*     KeySize - Key size requested.
*
* Return value
*     == 0 - No hardware API for the given key size.
*     != 0 - Hardware API for the given key size.
*/
static const CRYPTO_CIPHER_API *_GetHWAPI(unsigned KeySize) {
    const CRYPTO_CIPHER_API *pHWAPI;
    const CRYPTO_CIPHER_API *pSWAPI;
    const CRYPTO_CIPHER_API *pChosen;
    //
pChosen = 0;
CRYPTO_AES_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI != &CRYPTO_CIPHER_AES_SW) {
    pChosen = pHWAPI->pfClaim(KeySize);
    if (pChosen) {
        pHWAPI ->pfUnclaim(0);
    }
}
return pChosen;
}

```

```
*****
*
*      Public code
*
*****
*/
/* ****
*
*      MainTask()
*
*  Function description
*  Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_CIPHER_API * pAssist;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("AES Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz\n",
        SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION           = %u\n",
    [%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_AES_OPTIMIZE = %d\n",
    \n", CRYPTO_CONFIG_AES_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_AES_HW_OPTIMIZE = %d\n",
    \n", CRYPTO_CONFIG_AES_HW_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_CONFIG_GCM_OPTIMIZE = %d\n",
    \n", CRYPTO_CONFIG_GCM_OPTIMIZE);
    SEGGER_SYS_IO_Printf("\n");
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    SEGGER_SYS_IO_Printf(" |          |          | ECB       MB/s | CBC       MB/s |\n");
    SEGGER_SYS_IO_Printf(" | Cipher   | Bits     | Enc      Dec | Enc      Dec |\n");
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    //
    _CipherBenchmark_ECB_CBC("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES128_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES128_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("AES (HW)", pAssist, CRYPTO_AES128_KEY_SIZE);
    }
    _CipherBenchmark_ECB_CBC("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES192_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES192_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("AES (HW)", pAssist, CRYPTO_AES192_KEY_SIZE);
    }
    _CipherBenchmark_ECB_CBC("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES256_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES256_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("AES (HW)", pAssist, CRYPTO_AES256_KEY_SIZE);
    }
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    SEGGER_SYS_IO_Printf(" |          |          | GCM       MB/s | CCM       MB/s |\n");
    SEGGER_SYS_IO_Printf(" | Cipher   | Bits     | Enc      Dec | Enc      Dec |\n");
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    //
    _CipherBenchmark_GCM_CCM("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES128_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES128_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("AES (HW)", pAssist, CRYPTO_AES128_KEY_SIZE);
    }
    _CipherBenchmark_GCM_CCM("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES192_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES192_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("AES (HW)", pAssist, CRYPTO_AES192_KEY_SIZE);
    }
    _CipherBenchmark_GCM_CCM("AES", &CRYPTO_CIPHER_AES_SW, CRYPTO_AES256_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_AES256_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("AES (HW)", pAssist, CRYPTO_AES256_KEY_SIZE);
    }
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
}
```

```
//  
SEGGER_SYS_IO_Printf("\nBenchmark complete\n");  
SEGGER_SYS_OS_PauseBeforeHalt();  
SEGGER_SYS_OS_Halt(0);  
}  
  
***** End of file *****
```

21.1.2 CRYPTO_Bench_DES.c

This application benchmarks the configured performance of DES and TripleDES. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.1.2.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
DES Benchmark compiled Mar 19 2018 16:30:48

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed                  = 200.000 MHz
Config: CRYPTO_VERSION                 = 22400 [2.24]
Config: CRYPTO_CONFIG_DES_OPTIMIZE    = 5

+-----+-----+-----+-----+
|           |     ECB     MB/s |     CBC     MB/s |
| Cipher   | Bits   Enc   Dec |   Enc   Dec |
+-----+-----+-----+-----+
| DES      | 64    1.62  1.57 | 2.02  1.87 |
| DES (HW) | 64    3.02  2.72 | 4.53  3.40 |
| DES      | 128   0.62  0.62 | 0.70  0.68 |
| DES (HW) | 128   2.85  2.58 | 4.14  3.07 |
| DES      | 192   0.62  0.62 | 0.70  0.68 |
| DES (HW) | 192   2.85  2.58 | 4.14  3.07 |
+-----+-----+-----+-----+
* Note: key sizes include parity bits

Benchmark complete
```

21.1.2.2 Complete listing

```
/*
*          (c) SEGGER Microcontroller GmbH
*          The Embedded Experts
*          www.segger.com
*****
----- END-OF-HEADER -----
File       : CRYPTO_Bench_DES.c
Purpose    : Benchmark DES implementation.

*/
/*
*      #include section
*
*****
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
*      Static const data
*
*****
*/
static const U8 _aKey[32] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
};
```

```

/*
 *      Static data
 */
static U8 _aTestMessage[65536] = { 0 };

/*
 *      Static code
 */
_CipherBenchmark_ECB_CBC()

* Function description
* Convert ticks to seconds.

* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().

* Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

/*
 *      _CipherBenchmark_ECB_CBC()
 *
* Function description
* Benchmarks a cipher implementation.
*
* Parameters
*   sAlgorithm - Cipher algorithm name.
*   pAPI       - Pointer to cipher API.
*   KeySize    - Cipher key size in bytes.
*/
static void _CipherBenchmark_ECB_CBC(const char *sAlgorithm, const CRYPTO_CIPHER_API *pAPI, unsigned KeySize,
                                     CRYPTO_TDES_CONTEXT Context;
                                     U8          aIV[16];
                                     U64         T0;
                                     U64         OneSecond;
                                     unsigned     n;
                                     //SEGGER_SYS_IO_Printf(" | %-12s | %4d | ", sAlgorithm, KeySize*8);
                                     OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
                                     //
                                     T0 = SEGGER_SYS_OS_GetTimer();
                                     n = 0;
                                     pAPI->pfInitEncrypt(&Context, _aKey, KeySize);
                                     while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
                                         pAPI->pfEncrypt(&Context, &_aTestMessage[0], &_aTestMessage[0]);
                                         n += pAPI->BlockSize;
                                     }
                                     T0 = SEGGER_SYS_OS_GetTimer() - T0;
                                     SEGGER_SYS_IO_Printf("%6.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
                                     //
                                     T0 = SEGGER_SYS_OS_GetTimer();
                                     n = 0;
                                     pAPI->pfInitDecrypt(&Context, _aKey, KeySize);
                                     while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
                                         pAPI->pfDecrypt(&Context, &_aTestMessage[0], &_aTestMessage[0]);
                                         n += pAPI->BlockSize;
                                     }
                                     T0 = SEGGER_SYS_OS_GetTimer() - T0;
                                     SEGGER_SYS_IO_Printf("%6.2f |
                                     ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
                                     //

```

```

// CBC encrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, _aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_CIPHER_CBC_Encrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%6.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CBC decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitDecrypt(&Context, _aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_CIPHER_CBC_Decrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%6.2f |
\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****_GetTDESHardwareAssist()
*
* Function description
* Returns hardware acceleration API for given key size.
*
* Parameters
* KeySize - Key size requested.
*
* Return value
* == 0 - No hardware API for the given key size.
* != 0 - Hardware API for the given key size.
*/
static const CRYPTO_CIPHER_API *_GetTDESHardwareAssist(unsigned KeySize) {
    const CRYPTO_CIPHER_API *pHWAPI;
    const CRYPTO_CIPHER_API *pSWAPI;
    const CRYPTO_CIPHER_API *pChosen;
    //
    pChosen = 0;
    CRYPTO_TDES_QueryInstall(&pHWAPI, &pSWAPI);
    if (pHWAPI != &CRYPTO_CIPHER_TDES_SW) {
        pChosen = pHWAPI->pfClaim(KeySize);
        if (pChosen) {
            pHWAPI->pfUnclaim(0);
        }
    }
    return pChosen;
}

*****Public code
*****
*/
*****MainTask()
*
* Function description
* Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_CIPHER_API * pAssist;
    //

```

```

CRYPTO_Init();
SEGGER_SYS_Init();
//
SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
SEGGER_SYS_IO_Printf("DES Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
//
SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
if (SEGGER_SYS_GetProcessorSpeed() > 0) {
    SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz
\n", SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
}
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u
[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_DES_OPTIMIZE = %d\n",
CRYPTO_CONFIG_DES_OPTIMIZE);
SEGGER_SYS_IO_Printf("\n");
//
SEGGER_SYS_IO_Printf("-----+-----+-----+\n");
SEGGER_SYS_IO_Printf(" |           |           | ECB      MB/s | CBC      MB/s |\n");
SEGGER_SYS_IO_Printf(" | Cipher     | Bits       | Enc      Dec | Enc      Dec |\n");
SEGGER_SYS_IO_Printf("-----+-----+-----+\n");
//
_CipherBenchmark_ECB_CBC("DES", &CRYPTO_CIPHER_TDES_SW, CRYPTO_TDES_1KEY_SIZE);
if ((pAssist = _GetTDESHardwareAssist(CRYPTO_TDES_1KEY_SIZE)) != 0) {
    _CipherBenchmark_ECB_CBC("DES (HW)", pAssist, CRYPTO_TDES_1KEY_SIZE);
}
_CipherBenchmark_ECB_CBC("DES", &CRYPTO_CIPHER_TDES_SW, CRYPTO_TDES_2KEY_SIZE);
if ((pAssist = _GetTDESHardwareAssist(CRYPTO_TDES_2KEY_SIZE)) != 0) {
    _CipherBenchmark_ECB_CBC("DES (HW)", pAssist, CRYPTO_TDES_2KEY_SIZE);
}
_CipherBenchmark_ECB_CBC("DES", &CRYPTO_CIPHER_TDES_SW, CRYPTO_TDES_3KEY_SIZE);
if ((pAssist = _GetTDESHardwareAssist(CRYPTO_TDES_3KEY_SIZE)) != 0) {
    _CipherBenchmark_ECB_CBC("DES (HW)", pAssist, CRYPTO_TDES_3KEY_SIZE);
}
SEGGER_SYS_IO_Printf("-----+-----+-----+\n");
//
SEGGER_SYS_IO_Printf("\n* Note: key sizes include parity bits\n");
//
SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****/

```

21.1.3 CRYPTO_Bench_Camellia.c

This application benchmarks the configured performance of Camellia. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.1.3.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
Camellia Benchmark compiled Mar 19 2018 16:30:19

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION           = 22400 [2.24]
Config: CRYPTO_CONFIG_CAMELLIA_OPTIMIZE = 3
Config: CRYPTO_CONFIG_GCM_OPTIMIZE   = 0

+-----+-----+-----+-----+
|     |     | ECB    MB/s | CBC    MB/s |
| Cipher | Bits | Enc   Dec | Enc   Dec |
+-----+-----+-----+-----+
| CAMELLIA | 128 | 4.30  4.23 | 2.95  2.87 |
| CAMELLIA | 192 | 2.90  2.89 | 2.32  2.22 |
| CAMELLIA | 256 | 2.90  2.89 | 2.32  2.22 |
+-----+-----+-----+-----+
|     |     | GCM    MB/s | CCM    MB/s |
| Cipher | Bits | Enc   Dec | Enc   Dec |
+-----+-----+-----+-----+
| CAMELLIA | 128 | 0.11  0.11 | 1.50  1.52 |
| CAMELLIA | 192 | 0.11  0.11 | 1.16  1.17 |
| CAMELLIA | 256 | 0.11  0.11 | 1.15  1.16 |
+-----+-----+-----+-----+

Benchmark complete
```

21.1.3.2 Complete listing

```
*****
*               (c) SEGGER Microcontroller GmbH
*                   The Embedded Experts
*                   www.segger.com
*****  

----- END-OF-HEADER -----  

File       : CRYPTO_Bench_Camellia.c
Purpose    : Benchmark Camellia implementation.  

*/  

*****  

*  

*      #include section  

*  

*****  

*/  

#include "CRYPTO.h"
#include "SEGGER_SYS.h"  

*****  

*  

*      Static data  

*  

*****  

*/  

static U8 _aTestMessage[1024] = { 0 };
static U8 _aAAD[13]           = { 0 };
```

```

/*
*
*      Static code
*
***** Static code *****
*/
/*
*      _ConvertTicksToSeconds()
*
*      Function description
*          Convert ticks to seconds.
*
*      Parameters
*          Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
*      Return value
*          Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

/*
*      _CipherBenchmark_ECB_CBC()
*
*      Function description
*          Benchmarks a cipher implementation.
*
*      Parameters
*          sAlgorithm - Cipher algorithm name.
*          pAPI       - Pointer to cipher API.
*          KeySize    - Cipher key size in bytes.
*/
static void _CipherBenchmark_ECB_CBC(const char *sAlgorithm, const CRYPTO_CIPHER_API *pAPI, unsigned KeySize)
{
    CRYPTO_CAMELLIA_CONTEXT Context;
    U64                      T0;
    U64                      OneSecond;
    U8                       aIV [16];
    U8                       aKey[32];
    unsigned                  n;
    //
    CRYPTO_MEMZAP(aIV, sizeof(aIV));
    CRYPTO_MEMZAP(aKey, sizeof(aKey));
    //
    SEGGER_SYS_IO_Printf(" | %-12s | %4d | ", sAlgorithm, KeySize*8);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    // ECB encrypt
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInitEncrypt(&Context, aKey, KeySize);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        CRYPTO_CIPHER_ECB_Encrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), pAPI);
        n += sizeof(_aTestMessage);
    }
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
    //
    // ECB decrypt
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInitDecrypt(&Context, aKey, KeySize);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        CRYPTO_CIPHER_ECB_Decrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), pAPI);
        n += sizeof(_aTestMessage);
    }
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%7.2f |
", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

```

```

// CBC encrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_CIPHER_CBC_Encrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CBC decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitDecrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_CIPHER_CBC_Decrypt(&Context, &_aTestMessage[0], &_aTestMessage[0], sizeof(_aTestMessage), aIV, pAPI
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f |
\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
*      _CipherBenchmark_GCM_CCM()
*
*  Function description
*      Benchmarks a cipher implementation.
*
*  Parameters
*      sAlgorithm - Cipher algorithm name.
*      pAPI       - Pointer to cipher API.
*      KeySize    - Cipher key size in bytes.
*/
static void _CipherBenchmark_GCM_CCM(const char *sAlgorithm, const CRYPTO_CIPHER_API *pAPI, unsigned KeySize)
CRYPTO_CAMELLIA_CONTEXT Context;
U64                      T0;
U64                      OneSecond;
U8                       aIV[16];
U8                       aKey[32];
U8                       aTag[16];
unsigned                  n;
//
CRYPTO_MEMZAP(aIV, sizeof(aIV));
CRYPTO_MEMZAP(aKey, sizeof(aKey));
//
SEGGER_SYS_IO_Printf(" | %-12s | %4d | ", sAlgorithm, KeySize*8);
OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
//
// GCM encrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

    CRYPTO_CAMELLIA_GCM_Encrypt(&Context, &_aTestMessage[0], &aTag[0], sizeof(aTag), &_aTestMessage[0], sizeof(_aTestMessage)
        n += sizeof(_aTestMessage);
    }
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// GCM decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
}

```

```

CRYPTO_CAMELLIA_GCM_Decrypt(&Context, &aTestMessage[0], &aTag[0], sizeof(aTag), &aTestMessage[0], sizeof(aTestMessage));
    n += sizeof(_aTestMessage);
}
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f |", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CCM encrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

CRYPTO_CAMELLIA_CCM_Encrypt(&Context, &aTestMessage[0], &aTag[0], 16, &aTestMessage[0], sizeof(_aTestMessage));
    n += sizeof(_aTestMessage);
}
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f ", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
//
// CCM decrypt
//
T0 = SEGGER_SYS_OS_GetTimer();
n = 0;
pAPI->pfInitEncrypt(&Context, aKey, KeySize);
while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {

CRYPTO_CAMELLIA_CCM_Decrypt(&Context, &aTestMessage[0], &aTag[0], 16, &aTestMessage[0], sizeof(_aTestMessage));
    n += sizeof(_aTestMessage);
}
T0 = SEGGER_SYS_OS_GetTimer() - T0;
SEGGER_SYS_IO_Printf("%7.2f |%n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
*     _GetHWAPI()
*
* Function description
*     Returns hardware acceleration API for given key size.
*
* Parameters
*     KeySize - Key size requested.
*
* Return value
*     == 0 - No hardware API for the given key size.
*     != 0 - Hardware API for the given key size.
*/
static const CRYPTO_CIPHER_API *_GetHWAPI(unsigned KeySize) {
    const CRYPTO_CIPHER_API *pHWAPI;
    const CRYPTO_CIPHER_API *pSWAPI;
    const CRYPTO_CIPHER_API *pChosen;
    //
    pChosen = 0;
    CRYPTO_CAMELLIA_QueryInstall(&pHWAPI, &pSWAPI);
    if (pHWAPI != &CRYPTO_CIPHER_CAMELLIA_SW) {
        pChosen = pHWAPI->pfClaim(KeySize);
        if (pChosen) {
            pHWAPI ->pfUnclaim(0);
        }
    }
    return pChosen;
}

*****
*
*     Public code
*
*****
*/



/*
*     MainTask()
*/

```

```

/*
 *  Function description
 *  Main entry point for application to run all the tests.
 */
void MainTask(void);
void MainTask(void) {
    const CRYPTO_CIPHER_API * pAssist;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("Camellia Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz\n", SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_CAMELLIA_OPTIMIZE = %d\n", CRYPTO_CONFIG_CAMELLIA_OPTIMIZE);
    SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_GCM_OPTIMIZE = %d\n", CRYPTO_CONFIG_GCM_OPTIMIZE);
    SEGGER_SYS_IO_Printf("\n");
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    SEGGER_SYS_IO_Printf(" |           |   ECB     MB/s |   CBC     MB/s |\n");
    SEGGER_SYS_IO_Printf(" | Cipher   | Bits | Enc   Dec | Enc   Dec |\n");
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    //
    _CipherBenchmark_ECB_CBC("CAMELLIA", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA128_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA128_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("CAMELLIA (HW)", pAssist, CRYPTO_CAMELLIA128_KEY_SIZE);
    }
    _CipherBenchmark_ECB_CBC("CAMELLIA", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA192_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA192_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("CAMELLIA (HW)", pAssist, CRYPTO_CAMELLIA192_KEY_SIZE);
    }
    _CipherBenchmark_ECB_CBC("CAMELLIA", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA256_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA256_KEY_SIZE)) != NULL) {
        _CipherBenchmark_ECB_CBC("CAMELLIA (HW)", pAssist, CRYPTO_CAMELLIA256_KEY_SIZE);
    }
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    SEGGER_SYS_IO_Printf(" |           |   GCM     MB/s |   CCM     MB/s |\n");
    SEGGER_SYS_IO_Printf(" | Cipher   | Bits | Enc   Dec | Enc   Dec |\n");
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    //
    _CipherBenchmark_GCM_CCM("Camellia", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA128_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA128_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("Camellia(HW)", pAssist, CRYPTO_CAMELLIA128_KEY_SIZE);
    }
    _CipherBenchmark_GCM_CCM("Camellia", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA192_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA192_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("Camellia (HW)", pAssist, CRYPTO_CAMELLIA192_KEY_SIZE);
    }
    _CipherBenchmark_GCM_CCM("Camellia", &CRYPTO_CIPHER_CAMELLIA_SW, CRYPTO_CAMELLIA256_KEY_SIZE);
    if ((pAssist = _GetHWAPI(CRYPTO_CAMELLIA256_KEY_SIZE)) != NULL) {
        _CipherBenchmark_GCM_CCM("Camellia (HW)", pAssist, CRYPTO_CAMELLIA256_KEY_SIZE);
    }
    //
    SEGGER_SYS_IO_Printf("-----+-----+-----+-----+\n");
    //
    SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
    SEGGER_SYS_OS_PauseBeforeHalt();
    SEGGER_SYS_OS_Halt(0);
}

***** End of file *****/

```

21.2 Hashing

21.2.1 CRYPTO_Bench_MD5.c

This application benchmarks the configured performance of MD5. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.2.1.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
MD5 Benchmark compiled Mar 19 2018 16:34:02

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_CONFIG_MD5_OPTIMIZE = 1
Config: CRYPTO_CONFIG_MD5_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| MD5      |    25.10 |
+-----+-----+

Benchmark complete
```

21.2.1.2 Complete listing

```
/*
 *           (c) SEGGER Microcontroller GmbH
 *           The Embedded Experts
 *           www.segger.com
 */

----- END-OF-HEADER -----


File       : CRYPTO_Bench_MD5.c
Purpose    : Benchmark MD5 implementation.

*/



/*
 *      #include section
 *
***** Static data
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
 *      Static code
 */
*****
```

```

/*
*     _ConvertTicksToSeconds()
*
* Function description
*     Convert ticks to seconds.
*
* Parameters
*     Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*     Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

*****
*
*     _HashBenchmark()
*
* Function description
*     Benchmarks a hash implementation.
*
* Parameters
*     sAlgorithm - Hash algorithm name.
*     pAPI       - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_MD5_CONTEXT C;
    U64                 T0;
    U64                 OneSecond;
    unsigned            n;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfClaim) {
        pAPI->pfClaim();
    }
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |
\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
*     Public code
*
*****
*/
*****
```

```

*
*     MainTask()
*
* Function description
*     Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API *pHWAPI;
    const CRYPTO_HASH_API *pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("MD5 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
}

```

```
//  
SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());  
if (SEGGER_SYS_GetProcessorSpeed() > 0) {  
    SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz  
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);  
}  
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u  
[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());  
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_MD5_OPTIMIZE = %d  
\n", CRYPTO_CONFIG_MD5_OPTIMIZE);  
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_MD5_HW_OPTIMIZE = %d\n", CRYPTO_CONFIG_MD5_HW_OPTIMIZE);  
//  
SEGGER_SYS_IO_Printf("-----+\n");  
SEGGER_SYS_IO_Printf(" | Algorithm | Hash MB/s |\n");  
SEGGER_SYS_IO_Printf("-----+\n");  
//  
_HashBenchmark("MD5", &CRYPTO_HASH_MD5_SW);  
CRYPTO_MD5_QueryInstall(&pHWAPI, &pSWAPI);  
if (pHWAPI && pHWAPI != &CRYPTO_HASH_MD5_SW) {  
    _HashBenchmark("MD5 (HW)", pHWAPI);  
}  
SEGGER_SYS_IO_Printf("-----+\n");  
//  
SEGGER_SYS_IO_Printf("\nBenchmark complete\n");  
SEGGER_SYS_OS_PauseBeforeHalt();  
SEGGER_SYS_OS_Halt(0);  
}  
***** End of file *****
```

21.2.2 CRYPTO_Bench_SHA256.c

This application benchmarks the configured performance of SHA-1. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.2.2.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
SHA-256 Benchmark compiled Mar 19 2018 16:23:21

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed = 200.000 MHz
Config: CRYPTO_VERSION = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA256_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA256_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-256 (SW) |     3.61 |
| SHA-256 (HW) |   112.94 |
+-----+-----+

Benchmark complete
```

21.2.2.2 Complete listing

```
/****************************************************************************
 * (c) SEGGER Microcontroller GmbH
 *       The Embedded Experts
 *       www.segger.com
 ****

----- END-OF-HEADER -----


File      : CRYPTO_Bench_SHA256.c
Purpose   : Benchmark SHA-256 implementation.

*/



/*****
*
*      #include section
*
*****


#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*****
*
*      Static data
*
*****


static const U8 _aTestMessage[8192] = { 0 };

/*****
*
*      Static code
*
*****


/_ConvertTicksToSeconds()
```

```

/*
 * Function description
 * Convert ticks to seconds.
 *
 * Parameters
 * Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
 *
 * Return value
 * Number of seconds corresponding to tick.
 */
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

*****
*
* _HashBenchmark()
*
* Function description
* Benchmarks a hash implementation.
*
* Parameters
* sAlgorithm - Hash algorithm name.
* pAPI - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA256_CONTEXT C;
    U64 T0;
    U64 OneSecond;
    unsigned n;
    //
    SEGGER_SYS_IO_Printf("| %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfClaim) {
        pAPI->pfClaim();
    }
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
* Public code
*
*****
*/
*****
```

```

*
* MainTask()
*
* Function description
* Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API * pHWAPI;
    const CRYPTO_HASH_API * pSWAPI;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("SHA-256 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
```

```
SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
}
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u [%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA256_OPTIMIZE = %d\n", CRYPTO_CONFIG_SHA256_OPTIMIZE);
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA256_HW_OPTIMIZE = %d\n", CRYPTO_CONFIG_SHA256_HW_OPTIMIZE);
// SEGGER_SYS_IO_Printf("+-----+-----+\n");
SEGGER_SYS_IO_Printf(" | Algorithm | Hash MB/s |\n");
SEGGER_SYS_IO_Printf("+-----+-----+\n");
// _HashBenchmark("SHA-224 (SW)", &CRYPTO_HASH_SHA224_SW);
CRYPTO_SHA224_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI && pHWAPI != &CRYPTO_HASH_SHA224_SW) {
    _HashBenchmark("SHA-224 (HW)", pHWAPI);
}
_HashBenchmark("SHA-256 (SW)", &CRYPTO_HASH_SHA256_SW);
CRYPTO_SHA256_QueryInstall(&pHWAPI, &pSWAPI);
if (pHWAPI && pHWAPI != &CRYPTO_HASH_SHA256_SW) {
    _HashBenchmark("SHA-256 (HW)", pHWAPI);
}
SEGGER_SYS_IO_Printf("+-----+-----+\n");
// SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****
```

21.2.3 CRYPTO_Bench_SHA512.c

This application benchmarks the configured performance of SHA-1. It will benchmark both the software and hardware implementations, if a hardware accelerator is installed.

21.2.3.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
SHA-512 Benchmark compiled Mar 19 2018 16:43:06

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed = 200.000 MHz
Config: CRYPTO_VERSION = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA512_OPTIMIZE = 2
Config: CRYPTO_CONFIG_SHA512_HW_OPTIMIZE = 1

+-----+-----+
| Algorithm | Hash MB/s |
+-----+-----+
| SHA-512 (SW) | 1.57 |
+-----+-----+

Benchmark complete
```

21.2.3.2 Complete listing

```
/****************************************************************************
*           (c) SEGGER Microcontroller GmbH          *
*           The Embedded Experts                  *
*           www.segger.com                      *
*****
----- END-OF-HEADER -----
File      : CRYPTO_Bench_SHA512.c
Purpose   : Benchmark SHA-512 implementation.

*/
/*
*      #include section
*
*****
*/
#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
*      Static const data
*
*****
*/
static const U8 _aTestMessage[65536] = { 0 };

/*
*      Static code
*
*****
*/
_ConvertTicksToSeconds()

*
*      Function description
```

```

*   Convert ticks to seconds.
*
* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

//*********************************************************************
*
*       _HashBenchmark()
*
* Function description
*   Benchmarks a hash implementation.
*
* Parameters
*   sAlgorithm - Hash algorithm name.
*   pAPI       - Pointer to hash API.
*/
static void _HashBenchmark(const char *sAlgorithm, const CRYPTO_HASH_API *pAPI) {
    CRYPTO_SHA512_CONTEXT C; // big enough for most things...
    U64                  T0;
    U64                  OneSecond;
    unsigned             n;
    //
    SEGGER_SYS_IO_Printf(" | %-12s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    pAPI->pfInit(&C);
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfAdd(&C, &_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    pAPI->pfKill(&C);
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%9.2f |
\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

//*********************************************************************
*
*       Public code
*
*****
```

```

*****
```

```

*       MainTask()
*
* Function description
*   Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    const CRYPTO_HASH_API * pHwApi;
    const CRYPTO_HASH_API * pSwApi;
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    SEGGER_SYS_IO_Printf("%s     www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("SHA-512 Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed           = %.3f MHz
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
}
```

```
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u\n[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());  
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA512_OPTIMIZE = %d\n\r\n", CRYPTO_CONFIG_SHA512_OPTIMIZE);  
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA512_HW_OPTIMIZE = %d\r\n", CRYPTO_CONFIG_SHA256_HW_OPTIMIZE);  
//  
SEGGER_SYS_IO_Printf("+-----+\r\n");  
SEGGER_SYS_IO_Printf(" | Algorithm | Hash MB/s |\r\n");  
SEGGER_SYS_IO_Printf("+-----+\r\n");  
//  
_HashBenchmark("SHA-512 (SW)", &CRYPTO_HASH_SHA512_SW);  
CRYPTO_SHA512_QueryInstall(&pHWAPI, &pSWAPI);  
if (pHWAPI != &CRYPTO_HASH_SHA512_SW) {  
    _HashBenchmark("SHA-512 (HW)", pHWAPI);  
}  
SEGGER_SYS_IO_Printf("+-----+\r\n");  
//  
SEGGER_SYS_IO_Printf("\r\nBenchmark complete\r\n");  
SEGGER_SYS_OS_PauseBeforeHalt();  
SEGGER_SYS_OS_Halt(0);  
}  
  
***** End of file *****
```

21.3 Public key

21.3.1 CRYPTO_Bench_ModExp.c

This application benchmarks modular exponentiation that is the foundation of the RSA encryption scheme. It benchmarks both public and private key operations with the private key in both standard and Chinese Remainder Theorem (CRT) forms; it also benchmarks the different implementation techniques for modular exponentiation and a selection of modulus sizes.

21.3.1.1 Example output

Copyright (c) 2014-2018 SEGGER Microcontroller GmbH www.segger.com								
Modular Exponentiation Benchmark compiled Mar 19 2018 16:34:08								
Compiler: clang 5.0.0 (tags/RELEASE_500/initial)								
System: Processor speed = 200.000 MHz								
Config: CRYPTO_VERSION = 22400 [2.24]								
Config: CRYPTO_MPI_BITS_PER_LIMB = 32								
Modular Arithmetic Performance								
=====								
CRT private key, exponent length = modulus length, all times in ms								
Modulus		1024 bits				2048 bits		
Algorithm	Time	x	Memory	x	Time	x	Memory	
Basic, fast	66.23	1.00x	700	1.00x	344.62	1.00x	1340	1.00x
Basic, ladder	92.37	0.72x	840	1.20x	521.02	0.66x	1608	1.20x
Basic, 2b, FW	61.19	1.08x	1260	1.80x	330.05	1.04x	2412	1.80x
Basic, 3b, FW	57.42	1.15x	1820	2.60x	314.66	1.10x	3484	2.60x
Basic, 4b, FW	55.32	1.20x	2940	4.20x	302.74	1.14x	5628	4.20x
Basic, 5b, FW	54.86	1.21x	5180	7.40x	297.03	1.16x	9916	7.40x
Basic, 6b, FW	56.34	1.18x	9660	13.80x	295.46	1.17x	18492	13.80x
Basic, 2b, RM	60.37	1.10x	1260	1.80x	326.99	1.05x	2412	1.80x
Basic, 3b, RM	56.92	1.16x	1540	2.20x	310.28	1.11x	2948	2.20x
Basic, 4b, RM	54.70	1.21x	2100	3.00x	298.72	1.15x	4020	3.00x
Basic, 5b, RM	53.66	1.23x	3220	4.60x	292.52	1.18x	6164	4.60x
Basic, 6b, RM	53.72	1.23x	5460	7.80x	288.74	1.19x	10452	7.80x
Barrett, fast	74.10	0.89x	980	1.40x	354.35	0.97x	1876	1.40x
Barrett, ladder	102.28	0.65x	1120	1.60x	523.32	0.66x	2144	1.60x
Barrett, 2b, FW	69.92	0.95x	1540	2.20x	335.55	1.03x	2948	2.20x
Barrett, 3b, FW	64.86	1.02x	2100	3.00x	319.49	1.08x	4020	3.00x
Barrett, 4b, FW	62.27	1.06x	3220	4.60x	307.26	1.12x	6164	4.60x
Barrett, 5b, FW	61.69	1.07x	5460	7.80x	300.44	1.15x	10452	7.80x
Barrett, 6b, FW	63.31	1.05x	9940	14.20x	298.94	1.15x	19028	14.20x
Barrett, 2b, RM	67.31	0.98x	1540	2.20x	332.35	1.04x	2948	2.20x
Barrett, 3b, RM	63.65	1.04x	1820	2.60x	314.24	1.10x	3484	2.60x
Barrett, 4b, RM	61.09	1.08x	2380	3.40x	301.78	1.14x	4556	3.40x
Barrett, 5b, RM	60.00	1.10x	3500	5.00x	295.11	1.17x	6700	5.00x
Barrett, 6b, RM	60.09	1.10x	5740	8.20x	290.90	1.18x	10988	8.20x
Montgomery, fast	35.25	1.88x	700	1.00x	196.23	1.76x	1340	1.00x
Montgomery, 2b, FW	34.99	1.89x	1260	1.80x	194.78	1.77x	2412	1.80x
Montgomery, 3b, FW	31.46	2.10x	1820	2.60x	173.35	1.99x	3484	2.60x
Montgomery, 4b, FW	29.98	2.21x	2940	4.20x	164.24	2.10x	5628	4.20x
Montgomery, 5b, FW	29.57	2.24x	5180	7.40x	160.07	2.15x	9916	7.40x
Montgomery, 6b, FW	30.40	2.18x	9660	13.80x	159.98	2.15x	18492	13.80x
Montgomery, 2b, RM	32.22	2.06x	1260	1.80x	179.33	1.92x	2412	1.80x
Montgomery, 3b, RM	30.60	2.16x	1540	2.20x	168.59	2.04x	2948	2.20x
Montgomery, 4b, RM	29.48	2.25x	2100	3.00x	161.59	2.13x	4020	3.00x
Montgomery, 5b, RM	29.04	2.28x	3220	4.60x	158.26	2.18x	6164	4.60x
Montgomery, 6b, RM	29.01	2.28x	5460	7.80x	156.07	2.21x	10452	7.80x
Configured	66.24	1.00x	700	1.00x	344.67	1.00x	1340	1.00x
Public key, exponent length = 17 bits, all times in ms								
Modulus		1024 bits				2048 bits		
Algorithm	Time	x	Memory	x	Time	x	Memory	
Basic, fast	1.88	1.00x	804	1.00x	6.32	1.00x	1572	1.00x

Basic, ladder	3.98	0.47x	1072	1.33x	13.52	0.47x	2096	1.33x
Basic, 2b, FW	2.23	0.84x	1876	2.33x	7.36	0.86x	3668	2.33x
Basic, 3b, FW	2.63	0.72x	2948	3.67x	8.77	0.72x	5764	3.67x
Basic, 4b, FW	3.80	0.50x	5092	6.33x	12.73	0.50x	9956	6.33x
Basic, 5b, FW	5.81	0.32x	9380	11.67x	19.53	0.32x	18340	11.67x
Basic, 6b, FW	9.72	0.19x	17956	22.33x	32.52	0.19x	35108	22.33x
<hr/>								
Basic, 2b, RM	2.13	0.88x	1876	2.33x	7.15	0.88x	3668	2.33x
Basic, 3b, RM	2.40	0.78x	2412	3.00x	8.06	0.78x	4716	3.00x
Basic, 4b, RM	2.94	0.64x	3484	4.33x	9.89	0.64x	6812	4.33x
Basic, 5b, RM	4.01	0.47x	5628	7.00x	13.49	0.47x	11004	7.00x
Basic, 6b, RM	6.15	0.31x	9916	12.33x	20.67	0.31x	19388	12.33x
<hr/>								
Barrett, fast	2.14	0.88x	1340	1.67x	6.55	0.97x	2620	1.67x
Barrett, ladder	4.45	0.42x	1608	2.00x	13.93	0.45x	3144	2.00x
Barrett, 2b, FW	2.62	0.72x	2412	3.00x	7.75	0.82x	4716	3.00x
Barrett, 3b, FW	3.00	0.63x	3484	4.33x	9.07	0.70x	6812	4.33x
Barrett, 4b, FW	4.23	0.45x	5628	7.00x	12.95	0.49x	11004	7.00x
Barrett, 5b, FW	6.35	0.30x	9916	12.33x	19.63	0.32x	19388	12.33x
Barrett, 6b, FW	10.47	0.18x	18492	23.00x	32.46	0.19x	36156	23.00x
<hr/>								
Barrett, 2b, RM	2.40	0.78x	2412	3.00x	7.35	0.86x	4716	3.00x
Barrett, 3b, RM	2.68	0.70x	2948	3.67x	8.24	0.77x	5764	3.67x
Barrett, 4b, RM	3.26	0.58x	4020	5.00x	10.04	0.63x	7860	5.00x
Barrett, 5b, RM	4.39	0.43x	6164	7.67x	13.61	0.46x	12052	7.67x
Barrett, 6b, RM	6.64	0.28x	10452	13.00x	20.66	0.31x	20436	13.00x
<hr/>								
Montgomery, fast	1.40	1.35x	804	1.00x	4.55	1.39x	1572	1.00x
Montgomery, 2b, FW	2.08	0.91x	1876	2.33x	6.83	0.93x	3668	2.33x
Montgomery, 3b, FW	2.15	0.87x	2948	3.67x	7.05	0.90x	5764	3.67x
Montgomery, 4b, FW	2.74	0.69x	5092	6.33x	8.95	0.71x	9956	6.33x
Montgomery, 5b, FW	3.72	0.51x	9380	11.67x	12.13	0.52x	18340	11.67x
Montgomery, 6b, FW	5.61	0.34x	17956	22.33x	18.24	0.35x	35108	22.33x
<hr/>								
Montgomery, 2b, RM	1.59	1.18x	1876	2.33x	5.18	1.22x	3668	2.33x
Montgomery, 3b, RM	1.72	1.10x	2412	3.00x	5.61	1.13x	4716	3.00x
Montgomery, 4b, RM	2.10	0.90x	3484	4.33x	6.87	0.92x	6812	4.33x
Montgomery, 5b, RM	2.63	0.72x	5628	7.00x	8.55	0.74x	11004	7.00x
Montgomery, 6b, RM	3.56	0.53x	9916	12.33x	11.53	0.55x	19388	12.33x
<hr/>								
Configured	1.88	1.00x	804	1.00x	6.32	1.00x	1572	1.00x
<hr/>								

Benchmark complete

21.3.1.2 Complete listing

```
/*
 * (c) SEGGER Microcontroller GmbH
 *      The Embedded Experts
 *      www.segger.com
 */

----- END-OF-HEADER -----

File      : CRYPTO_Bench_ModExp.c
Purpose   : Benchmark modular exponentiation.

*/
/*
 *      #include section
 *
*****
 */
#include "CRYPTO.h"
#include "SEGGER_MEM.h"
#include "SEGGER_SYS.h"

/*
 *      Defines, configurable
 *
*****
 */
#define INCLUDE_SMALL_MODULI 0
```

```

#define INCLUDE_PLAIN_PRIVATE 0           // Include plain private key operations
[not really used in practice]
#define INCLUDE_EFM32 0
// Include EFM32 benchmarks using CRYPTO unit
#define MAX_CHUNKS 75

//*****************************************************************************
*
*      Defines, fixed
*
*****
*/
#define CRYPTO_ASSERT(X)           { if (!(X)) { CRYPTO_PANIC(); } }
// I know this is low-rent
#define CRYPTO_CHECK(X)           /*lint -e{717,801,9036}
/* do { if ((Status = (X)) < 0) goto Finally; } while (0)

//*****************************************************************************
*
*      Defines, fixed
*
*****
*/
#if INCLUDE_EFM32
#define EFM32(X, Y, Z) X, Y, Z
#else
#define EFM32(X, Y, Z)
#endif

//*****************************************************************************
*
*      Local data types
*
*****
*/
#if INCLUDE_EFM32
typedef CRYPTO_MPI_LIMB MPI_UNIT[CRYPTO_MPI_LIMBS_REQUIRED(2*2048)+4*128/32]; // 128-bit megadigit implementation
#else
typedef CRYPTO_MPI_LIMB MPI_UNIT[CRYPTO_MPI_LIMBS_REQUIRED(2*2048)+4];
#endif

typedef int (*MODEXP_FUNC)
(CRYPTO_MPI *pSelf, const CRYPTO_MPI *pExponent, const CRYPTO_MPI *pModulus, SEGGER_MEM_CONTEXT *pMem);

typedef struct {
    const char * pText;           // Description of algorithm
    MODEXP_FUNC pfModExp;
} BENCH_ALG;

typedef struct {
    const CRYPTO_MPI *pN;
    const CRYPTO_MPI *pE;
    const CRYPTO_MPI *pP;
    const CRYPTO_MPI *pQ;
    const CRYPTO_MPI *pDP;
    const CRYPTO_MPI *pDQ;
    const CRYPTO_MPI *pQInv;
} BENCH_KEY;

typedef void (*BENCH_FUNC)(MODEXP_FUNC pfModExp, const BENCH_KEY *pKey);

typedef struct {
    const char * pText;           // Description of scenario
    BENCH_FUNC pfBenchFunc;
} BENCH_SCENARIO;

//*****************************************************************************
*
*      Prototypes
*
*****
*/

```

```

#if INCLUDE_PLAIN_PRIVATE
static void _BenchmarkModExp_Private_Plain(MODEXP_FUNC pfModExp, const BENCH_KEY *pKey);
#endif
static void _BenchmarkModExp_Private_CRT (MODEXP_FUNC pfModExp, const BENCH_KEY * pKey);
static void _BenchmarkModExp_Public (MODEXP_FUNC pfModExp, const BENCH_KEY *pKey);

/*****************
*      Static const data
*
***** */

#endif

#if INCLUDE_SMALL_MODULI

/*****************
*      128-bit modulus
*/
__MPI_LITERAL_BEGIN(static, _RSA128_N)
__MPI_LITERAL_DATA(0x37, 0x28, 0x3b, 0x11),
__MPI_LITERAL_DATA(0x68, 0x8d, 0xe5, 0x7c),
__MPI_LITERAL_DATA(0x47, 0x41, 0x65, 0x41),
__MPI_LITERAL_DATA(0x22, 0x25, 0xdf, 0xb2)
__MPI_LITERAL_END(static, _RSA128_N, 128)

__MPI_LITERAL_BEGIN(static, _RSA128_E)
__MPI_LITERAL_DATA(0x01, 0x00, 0x01, 0x00)
__MPI_LITERAL_END(static, _RSA128_E, 17)

__MPI_LITERAL_BEGIN(static, _RSA128_P)
__MPI_LITERAL_DATA(0x63, 0x30, 0x6a, 0x78),
__MPI_LITERAL_DATA(0x5e, 0x2c, 0xf8, 0xc2)
__MPI_LITERAL_END(static, _RSA128_P, 64)

__MPI_LITERAL_BEGIN(static, _RSA128_Q)
__MPI_LITERAL_DATA(0x1d, 0xaf, 0x60, 0xdb),
__MPI_LITERAL_DATA(0x2f, 0xfb, 0xdc, 0xea)
__MPI_LITERAL_END(static, _RSA128_Q, 64)

__MPI_LITERAL_BEGIN(static, _RSA128_DP)
__MPI_LITERAL_DATA(0xd1, 0xb2, 0x3b, 0x5f),
__MPI_LITERAL_DATA(0x62, 0x33, 0x5f, 0xc1)
__MPI_LITERAL_END(static, _RSA128_DP, 64)

__MPI_LITERAL_BEGIN(static, _RSA128_DQ)
__MPI_LITERAL_DATA(0x85, 0x66, 0x35, 0x1e),
__MPI_LITERAL_DATA(0xe3, 0xfc, 0xd2, 0x74)
__MPI_LITERAL_END(static, _RSA128_DQ, 63)

__MPI_LITERAL_BEGIN(static, _RSA128_QINV)
__MPI_LITERAL_DATA(0x38, 0xf1, 0xdc, 0x71),
__MPI_LITERAL_DATA(0x4e, 0xa, 0xec, 0xa4)
__MPI_LITERAL_END(static, _RSA128_QINV, 64)

/*****************
*      256-bit modulus
*/
__MPI_LITERAL_BEGIN(static, _RSA256_N)
__MPI_LITERAL_DATA(0xef, 0xcc, 0xb1, 0x30),
__MPI_LITERAL_DATA(0xc7, 0x06, 0x36, 0xaf),
__MPI_LITERAL_DATA(0xe7, 0x14, 0xc9, 0x14),
__MPI_LITERAL_DATA(0x3c, 0xc0, 0x0f, 0x48),
__MPI_LITERAL_DATA(0x4b, 0xc6, 0xa6, 0xab),
__MPI_LITERAL_DATA(0xb3, 0x62, 0xbb, 0x52),
__MPI_LITERAL_DATA(0xa4, 0xa0, 0xbb, 0x1a),
__MPI_LITERAL_DATA(0x26, 0xcf, 0xb8, 0x9d)
__MPI_LITERAL_END(static, _RSA256_N, 256)

__MPI_LITERAL_BEGIN(static, _RSA256_E)
__MPI_LITERAL_DATA(0x01, 0x00, 0x01, 0x00)
__MPI_LITERAL_END(static, _RSA256_E, 17)

```

```

__MPI_LITERAL_BEGIN(static, _RSA256_P)
__MPI_LITERAL_DATA(0x73, 0x10, 0x31, 0xa2),
__MPI_LITERAL_DATA(0xfe, 0xd9, 0xf7, 0x90),
__MPI_LITERAL_DATA(0x53, 0x2f, 0x0e, 0x4b),
__MPI_LITERAL_DATA(0x9d, 0x50, 0xa9, 0xc3)
__MPI_LITERAL_END(static, _RSA256_P, 128)

__MPI_LITERAL_BEGIN(static, _RSA256_Q)
__MPI_LITERAL_DATA(0x95, 0x5e, 0x43, 0xa8),
__MPI_LITERAL_DATA(0x96, 0xef, 0x23, 0x3a),
__MPI_LITERAL_DATA(0x3a, 0xd3, 0x69, 0x41),
__MPI_LITERAL_DATA(0xfe, 0x52, 0x5c, 0xce)
__MPI_LITERAL_END(static, _RSA256_Q, 128)

__MPI_LITERAL_BEGIN(static, _RSA256_DP)
__MPI_LITERAL_DATA(0x99, 0x7c, 0x4c, 0x04),
__MPI_LITERAL_DATA(0xe8, 0x1f, 0xcd, 0x61),
__MPI_LITERAL_DATA(0xc8, 0x5e, 0x98, 0xcc),
__MPI_LITERAL_DATA(0xcb, 0xbd, 0x0e, 0xbe)
__MPI_LITERAL_END(static, _RSA256_DP, 128)

__MPI_LITERAL_BEGIN(static, _RSA256_DQ)
__MPI_LITERAL_DATA(0x39, 0xb0, 0x00, 0x8e),
__MPI_LITERAL_DATA(0xa4, 0x1e, 0x06, 0xa5),
__MPI_LITERAL_DATA(0xfe, 0xe6, 0x55, 0x09),
__MPI_LITERAL_DATA(0x70, 0x1f, 0xa8, 0x0b)
__MPI_LITERAL_END(static, _RSA256_DQ, 124)

__MPI_LITERAL_BEGIN(static, _RSA256_QINV)
__MPI_LITERAL_DATA(0x82, 0x19, 0xf4, 0x2b),
__MPI_LITERAL_DATA(0xea, 0x4f, 0xel, 0xa8),
__MPI_LITERAL_DATA(0x4b, 0x3b, 0x3f, 0xb1),
__MPI_LITERAL_DATA(0xe0, 0xc1, 0xb8, 0x94)
__MPI_LITERAL_END(static, _RSA256_QINV, 128)

//*****************************************************************************
*
*      512-bit modulus
*/
__MPI_LITERAL_BEGIN(static, _RSA512_N)
__MPI_LITERAL_DATA(0x59, 0xae, 0x18, 0x11),
__MPI_LITERAL_DATA(0xc4, 0x8a, 0xe4, 0x73),
__MPI_LITERAL_DATA(0x24, 0xfd, 0xf3, 0x08),
__MPI_LITERAL_DATA(0x40, 0x9b, 0x6b, 0x4e),
__MPI_LITERAL_DATA(0x07, 0x01, 0x94, 0x87),
__MPI_LITERAL_DATA(0xd8, 0xbf, 0x28, 0x45),
__MPI_LITERAL_DATA(0x85, 0x8b, 0x65, 0x10),
__MPI_LITERAL_DATA(0x8f, 0x82, 0x8a, 0x38),
__MPI_LITERAL_DATA(0x12, 0x6a, 0xb1, 0x48),
__MPI_LITERAL_DATA(0x09, 0x44, 0xf5, 0xd4),
__MPI_LITERAL_DATA(0xa9, 0x62, 0x76, 0xd2),
__MPI_LITERAL_DATA(0x5b, 0xa5, 0x10, 0x15),
__MPI_LITERAL_DATA(0x9b, 0xa5, 0xa6, 0x36),
__MPI_LITERAL_DATA(0xf8, 0x8e, 0xbe, 0x5b),
__MPI_LITERAL_DATA(0x17, 0x59, 0x63, 0x44),
__MPI_LITERAL_DATA(0xe4, 0x23, 0x35, 0xbe)
__MPI_LITERAL_END(static, _RSA512_N, 512)

__MPI_LITERAL_BEGIN(static, _RSA512_E)
__MPI_LITERAL_DATA(0x01, 0x00, 0x01, 0x00)
__MPI_LITERAL_END(static, _RSA512_E, 17)

__MPI_LITERAL_BEGIN(static, _RSA512_P)
__MPI_LITERAL_DATA(0xc1, 0x81, 0x46, 0x93),
__MPI_LITERAL_DATA(0xf4, 0x07, 0x58, 0xcb),
__MPI_LITERAL_DATA(0x90, 0x25, 0x97, 0x29),
__MPI_LITERAL_DATA(0x65, 0x38, 0x27, 0x35),
__MPI_LITERAL_DATA(0xf4, 0xe3, 0xd6, 0x51),
__MPI_LITERAL_DATA(0x0e, 0xf5, 0x88, 0x7d),
__MPI_LITERAL_DATA(0xda, 0x87, 0x3f, 0xd0),
__MPI_LITERAL_DATA(0x28, 0x43, 0x6a, 0xc9)
__MPI_LITERAL_END(static, _RSA512_P, 256)

__MPI_LITERAL_BEGIN(static, _RSA512_Q)

```

```

__MPI_LITERAL_DATA(0x99, 0xa2, 0x19, 0x42),
__MPI_LITERAL_DATA(0xcf, 0x0a, 0x50, 0xbff),
__MPI_LITERAL_DATA(0xec, 0x20, 0x03, 0x5b),
__MPI_LITERAL_DATA(0x7b, 0x90, 0x11, 0x4c),
__MPI_LITERAL_DATA(0x92, 0x18, 0xe9, 0x1a),
__MPI_LITERAL_DATA(0xbd, 0x67, 0x2b, 0x3e),
__MPI_LITERAL_DATA(0xd0, 0x02, 0xc5, 0x4f),
__MPI_LITERAL_DATA(0x52, 0x53, 0xc1, 0xf1)
__MPI_LITERAL_END(static, _RSA512_Q, 256)

__MPI_LITERAL_BEGIN(static, _RSA512_DP)
__MPI_LITERAL_DATA(0x41, 0xbc, 0x3a, 0xac),
__MPI_LITERAL_DATA(0xd6, 0x34, 0x12, 0x0f),
__MPI_LITERAL_DATA(0x19, 0x92, 0x7d, 0x5c),
__MPI_LITERAL_DATA(0x85, 0xde, 0x65, 0xe4),
__MPI_LITERAL_DATA(0xb9, 0xb3, 0xff, 0x6e),
__MPI_LITERAL_DATA(0xcf, 0x38, 0x3d, 0x68),
__MPI_LITERAL_DATA(0xd8, 0xa5, 0xaf, 0xf9),
__MPI_LITERAL_DATA(0xa5, 0xf8, 0xa6, 0xa4)
__MPI_LITERAL_END(static, _RSA512_DP, 256)

__MPI_LITERAL_BEGIN(static, _RSA512_DQ)
__MPI_LITERAL_DATA(0x21, 0xf0, 0x5f, 0x76),
__MPI_LITERAL_DATA(0x2c, 0x1b, 0x07, 0x6f),
__MPI_LITERAL_DATA(0x51, 0x8f, 0x81, 0x1e),
__MPI_LITERAL_DATA(0xdd, 0x6d, 0xce, 0xdd),
__MPI_LITERAL_DATA(0x1c, 0x81, 0x4d, 0x74),
__MPI_LITERAL_DATA(0x83, 0xdf, 0x58, 0x28),
__MPI_LITERAL_DATA(0x01, 0x71, 0x43, 0x04),
__MPI_LITERAL_DATA(0x2e, 0xe5, 0x8d, 0x63)
__MPI_LITERAL_END(static, _RSA512_DQ, 255)

__MPI_LITERAL_BEGIN(static, _RSA512_QINV)
__MPI_LITERAL_DATA(0x6f, 0xc3, 0x88, 0x72),
__MPI_LITERAL_DATA(0x72, 0x2b, 0x9f, 0x29),
__MPI_LITERAL_DATA(0x02, 0x27, 0x41, 0x6f),
__MPI_LITERAL_DATA(0xdb, 0xd5, 0xad, 0xb0),
__MPI_LITERAL_DATA(0x66, 0xd3, 0x16, 0x67),
__MPI_LITERAL_DATA(0xdc, 0x31, 0x0c, 0x61),
__MPI_LITERAL_DATA(0x07, 0xfb, 0x2f, 0x88),
__MPI_LITERAL_DATA(0x2d, 0x86, 0xef, 0x6b)
__MPI_LITERAL_END(static, _RSA512_QINV, 255)

#endif

/********************* 1024-bit modulus *****/
* / 1024-bit modulus */

__MPI_LITERAL_BEGIN(static, _RSA1024_N)
__MPI_LITERAL_DATA(0x69, 0x79, 0xab, 0x83),
__MPI_LITERAL_DATA(0x84, 0x03, 0x2a, 0x64),
__MPI_LITERAL_DATA(0xbb, 0x79, 0x87, 0xf9),
__MPI_LITERAL_DATA(0x89, 0x56, 0x97, 0x96),
__MPI_LITERAL_DATA(0xcc, 0x8c, 0x6f, 0xe2),
__MPI_LITERAL_DATA(0x86, 0xa9, 0xdf, 0x09),
__MPI_LITERAL_DATA(0x11, 0x1e, 0x4c, 0x9c),
__MPI_LITERAL_DATA(0xf9, 0x47, 0xf1, 0xe1),
__MPI_LITERAL_DATA(0x96, 0x0b, 0x06, 0xfe),
__MPI_LITERAL_DATA(0xcc, 0x59, 0xcd, 0x24),
__MPI_LITERAL_DATA(0x08, 0x1c, 0xd6, 0x18),
__MPI_LITERAL_DATA(0x64, 0xee, 0xaa, 0x8b),
__MPI_LITERAL_DATA(0x42, 0xb6, 0x7a, 0x80),
__MPI_LITERAL_DATA(0x76, 0xee, 0x77, 0xc5),
__MPI_LITERAL_DATA(0x57, 0x4b, 0x7e, 0x04),
__MPI_LITERAL_DATA(0x83, 0xc0, 0xf4, 0x96),
__MPI_LITERAL_DATA(0x20, 0x53, 0x39, 0x77),
__MPI_LITERAL_DATA(0xa0, 0x7a, 0x74, 0x36),
__MPI_LITERAL_DATA(0x07, 0x25, 0x44, 0xf3),
__MPI_LITERAL_DATA(0x6e, 0x85, 0x8c, 0x01),
__MPI_LITERAL_DATA(0xc2, 0x29, 0x6f, 0xcc),
__MPI_LITERAL_DATA(0x48, 0x18, 0xad, 0xc6),
__MPI_LITERAL_DATA(0x86, 0x6d, 0xcb, 0x3e),
__MPI_LITERAL_DATA(0x12, 0x49, 0x53, 0xb6),
__MPI_LITERAL_DATA(0x26, 0x25, 0xb9, 0xc9),

```

```

__MPI_LITERAL_DATA(0x8c, 0x3b, 0xec, 0x27),
__MPI_LITERAL_DATA(0x7e, 0xc0, 0x7c, 0x4a),
__MPI_LITERAL_DATA(0x27, 0xad, 0xa, 0x64),
__MPI_LITERAL_DATA(0xf6, 0xd4, 0x5d, 0x4b),
__MPI_LITERAL_DATA(0xa0, 0xf1, 0x46, 0x96),
__MPI_LITERAL_DATA(0xcc, 0xc1, 0xc9, 0x0f),
__MPI_LITERAL_DATA(0x4e, 0xb4, 0x5b, 0xca)
__MPI_LITERAL_END(static, _RSA1024_N, 1024)

__MPI_LITERAL_BEGIN(static, _RSA1024_E)
__MPI_LITERAL_DATA(0x01, 0x00, 0x01, 0x00)
__MPI_LITERAL_END(static, _RSA1024_E, 17)

__MPI_LITERAL_BEGIN(static, _RSA1024_P)
__MPI_LITERAL_DATA(0x19, 0x63, 0x9c, 0xf6),
__MPI_LITERAL_DATA(0xc5, 0x2f, 0x5a, 0x80),
__MPI_LITERAL_DATA(0xa7, 0x8c, 0x2a, 0x53),
__MPI_LITERAL_DATA(0x4a, 0x1d, 0x7b, 0x34),
__MPI_LITERAL_DATA(0x9d, 0x0d, 0x99, 0xfb),
__MPI_LITERAL_DATA(0x8f, 0x74, 0xa2, 0x28),
__MPI_LITERAL_DATA(0x96, 0x50, 0x5f, 0x55),
__MPI_LITERAL_DATA(0x42, 0xe7, 0xb5, 0x3b),
__MPI_LITERAL_DATA(0x9e, 0x91, 0xb2, 0x8a),
__MPI_LITERAL_DATA(0x1d, 0xca, 0xf2, 0x5a),
__MPI_LITERAL_DATA(0xbb, 0xbc, 0x15, 0xe8),
__MPI_LITERAL_DATA(0xde, 0x2b, 0x58, 0x35),
__MPI_LITERAL_DATA(0x38, 0xbf, 0xe7, 0x3c),
__MPI_LITERAL_DATA(0x22, 0x00, 0xd9, 0x7b),
__MPI_LITERAL_DATA(0xe0, 0xaf, 0xf9, 0xb4),
__MPI_LITERAL_DATA(0x02, 0x2e, 0x8d, 0xd0)
__MPI_LITERAL_END(static, _RSA1024_P, 512)

__MPI_LITERAL_BEGIN(static, _RSA1024_Q)
__MPI_LITERAL_DATA(0xd1, 0x62, 0x67, 0x19),
__MPI_LITERAL_DATA(0xad, 0x64, 0xf7, 0xe3),
__MPI_LITERAL_DATA(0xf6, 0x77, 0x04, 0xcd),
__MPI_LITERAL_DATA(0x83, 0xef, 0xf3, 0x4f),
__MPI_LITERAL_DATA(0xf2, 0x08, 0xc7, 0xeb),
__MPI_LITERAL_DATA(0xfc, 0x95, 0x3f, 0x0b),
__MPI_LITERAL_DATA(0x34, 0x06, 0x46, 0xf3),
__MPI_LITERAL_DATA(0x79, 0xad, 0xe3, 0xf8),
__MPI_LITERAL_DATA(0xa6, 0x11, 0x4b, 0x66),
__MPI_LITERAL_DATA(0xd3, 0x51, 0x3e, 0x0c),
__MPI_LITERAL_DATA(0xd, 0xa0, 0x28, 0xd8),
__MPI_LITERAL_DATA(0xf0, 0x38, 0x16, 0xa3),
__MPI_LITERAL_DATA(0x02, 0xaa, 0x2e, 0x4f),
__MPI_LITERAL_DATA(0x8e, 0xe9, 0xc9, 0x7b),
__MPI_LITERAL_DATA(0xd3, 0x33, 0x36, 0x1f),
__MPI_LITERAL_DATA(0x43, 0xce, 0x65, 0xf8)
__MPI_LITERAL_END(static, _RSA1024_Q, 512)

__MPI_LITERAL_BEGIN(static, _RSA1024_DP)
__MPI_LITERAL_DATA(0x59, 0x8c, 0x4f, 0xdf),
__MPI_LITERAL_DATA(0x08, 0xe0, 0xf2, 0xf2),
__MPI_LITERAL_DATA(0x32, 0x1d, 0x35, 0x49),
__MPI_LITERAL_DATA(0x8b, 0x5d, 0xe4, 0x25),
__MPI_LITERAL_DATA(0xe2, 0x21, 0xa8, 0xed),
__MPI_LITERAL_DATA(0x1e, 0xed, 0xf8, 0x65),
__MPI_LITERAL_DATA(0x49, 0x3e, 0xe3, 0x00),
__MPI_LITERAL_DATA(0xba, 0x4a, 0x93, 0x70),
__MPI_LITERAL_DATA(0x08, 0x88, 0x1a, 0x51),
__MPI_LITERAL_DATA(0x56, 0x48, 0x8a, 0x9f),
__MPI_LITERAL_DATA(0x24, 0xec, 0x1e, 0x14),
__MPI_LITERAL_DATA(0xd, 0x0f, 0x59, 0xfa),
__MPI_LITERAL_DATA(0xb0, 0x74, 0x0e, 0x8c),
__MPI_LITERAL_DATA(0x4a, 0x72, 0x90, 0xc7),
__MPI_LITERAL_DATA(0x75, 0x35, 0xd1, 0xbf),
__MPI_LITERAL_DATA(0x73, 0x48, 0xdf, 0x3c)
__MPI_LITERAL_END(static, _RSA1024_DP, 510)

__MPI_LITERAL_BEGIN(static, _RSA1024_DQ)
__MPI_LITERAL_DATA(0x11, 0x67, 0x61, 0xb7),
__MPI_LITERAL_DATA(0xf2, 0x78, 0x2c, 0xa1),
__MPI_LITERAL_DATA(0x52, 0x1c, 0xe4, 0xeb),
__MPI_LITERAL_DATA(0xfc, 0xa5, 0x8d, 0xdb),
__MPI_LITERAL_DATA(0xb2, 0xf3, 0xd0, 0x7a),

```

```

__MPI_LITERAL_DATA(0x7a, 0x3f, 0xd4, 0x72),
__MPI_LITERAL_DATA(0xad, 0x8d, 0xc3, 0x9b),
__MPI_LITERAL_DATA(0x06, 0x34, 0x35, 0xab),
__MPI_LITERAL_DATA(0xa2, 0x56, 0x68, 0x8c),
__MPI_LITERAL_DATA(0x60, 0x1b, 0xfb, 0x27),
__MPI_LITERAL_DATA(0x12, 0x02, 0x28, 0x4f),
__MPI_LITERAL_DATA(0x8c, 0xf8, 0xa8, 0xdb),
__MPI_LITERAL_DATA(0xfb, 0x38, 0x30, 0x24),
__MPI_LITERAL_DATA(0x17, 0xf0, 0x8c, 0x2e),
__MPI_LITERAL_DATA(0xb7, 0x98, 0x01, 0x5d),
__MPI_LITERAL_DATA(0xee, 0x1c, 0x3b, 0xf8)
__MPI_LITERAL_END(static, _RSA1024_DQ, 512)

__MPI_LITERAL_BEGIN(static, _RSA1024_QINV)
__MPI_LITERAL_DATA(0xd4, 0x63, 0x00, 0xbf),
__MPI_LITERAL_DATA(0xbc, 0x22, 0x79, 0x05),
__MPI_LITERAL_DATA(0x86, 0x76, 0x8b, 0x10),
__MPI_LITERAL_DATA(0xf9, 0xe3, 0x65, 0x64),
__MPI_LITERAL_DATA(0xbe, 0xff, 0x69, 0xef),
__MPI_LITERAL_DATA(0x9f, 0x2a, 0xa2, 0x7d),
__MPI_LITERAL_DATA(0x33, 0xe8, 0xac, 0x7b),
__MPI_LITERAL_DATA(0x25, 0x03, 0xf1, 0xa2),
__MPI_LITERAL_DATA(0x1b, 0x31, 0xa3, 0xd7),
__MPI_LITERAL_DATA(0x6f, 0xd4, 0xdf, 0x37),
__MPI_LITERAL_DATA(0x6a, 0x8c, 0x59, 0xab),
__MPI_LITERAL_DATA(0x19, 0x61, 0x3c, 0x62),
__MPI_LITERAL_DATA(0x20, 0x61, 0xf6, 0x41),
__MPI_LITERAL_DATA(0x48, 0x9e, 0xfd, 0x7a),
__MPI_LITERAL_DATA(0xc0, 0x21, 0x1b, 0xde),
__MPI_LITERAL_END(static, _RSA1024_QINV, 51)

/*****************
*      2048-bit modulus
*/
static const CRYPTO_MPI_LIMB _RSA2048_N_aLimbs[] = {
CRYPTO_MPI_LIMB_DATA4(0xe7, 0xfe, 0x60, 0x71),
CRYPTO_MPI_LIMB_DATA4(0x5f, 0x76, 0x91, 0xee),
CRYPTO_MPI_LIMB_DATA4(0x16, 0x60, 0x98, 0xa),
CRYPTO_MPI_LIMB_DATA4(0x5e, 0x71, 0x2f, 0x4f),
CRYPTO_MPI_LIMB_DATA4(0x17, 0x6b, 0x5d, 0x2a),
CRYPTO_MPI_LIMB_DATA4(0x22, 0xb2, 0x4f, 0x71),
CRYPTO_MPI_LIMB_DATA4(0x1d, 0xd1, 0xef, 0x56),
CRYPTO_MPI_LIMB_DATA4(0x06, 0xf8, 0xe7, 0x80),
CRYPTO_MPI_LIMB_DATA4(0xfb, 0xa5, 0x3, 0xc2),
CRYPTO_MPI_LIMB_DATA4(0x4b, 0xc0, 0xeb, 0xfa),
CRYPTO_MPI_LIMB_DATA4(0x51, 0x6d, 0x9e, 0xb9),
CRYPTO_MPI_LIMB_DATA4(0x1b, 0xce, 0xe3, 0x57),
CRYPTO_MPI_LIMB_DATA4(0x46, 0xa5, 0x3e, 0x32),
CRYPTO_MPI_LIMB_DATA4(0x3e, 0x12, 0x6e, 0xd8),
CRYPTO_MPI_LIMB_DATA4(0x5c, 0x84, 0x65, 0xce),
CRYPTO_MPI_LIMB_DATA4(0x31, 0xda, 0x2e, 0x80),
CRYPTO_MPI_LIMB_DATA4(0xae, 0xfc, 0xda, 0x17),
CRYPTO_MPI_LIMB_DATA4(0x35, 0x3e, 0xb1, 0xe8),
CRYPTO_MPI_LIMB_DATA4(0x48, 0xb9, 0x9f, 0xe1),
CRYPTO_MPI_LIMB_DATA4(0x51, 0xa6, 0xcc, 0xd2),
CRYPTO_MPI_LIMB_DATA4(0x3b, 0xa5, 0x1a, 0xf9),
CRYPTO_MPI_LIMB_DATA4(0x5c, 0x8d, 0x7c, 0x05),
CRYPTO_MPI_LIMB_DATA4(0x25, 0x06, 0x35, 0x15),
CRYPTO_MPI_LIMB_DATA4(0x26, 0x4e, 0x45, 0xd6),
CRYPTO_MPI_LIMB_DATA4(0x46, 0x33, 0x31, 0xd6),
CRYPTO_MPI_LIMB_DATA4(0x97, 0x18, 0xbe, 0x5b),
CRYPTO_MPI_LIMB_DATA4(0xa1, 0xfa, 0x39, 0x9a),
CRYPTO_MPI_LIMB_DATA4(0xe4, 0x4f, 0x82, 0xbe),
CRYPTO_MPI_LIMB_DATA4(0xfe, 0x99, 0xab, 0x5a),
CRYPTO_MPI_LIMB_DATA4(0xab, 0x33, 0xd9, 0xb6),
CRYPTO_MPI_LIMB_DATA4(0xa9, 0x4a, 0xf6, 0xa1),
CRYPTO_MPI_LIMB_DATA4(0x70, 0xa4, 0x1b, 0xae),
CRYPTO_MPI_LIMB_DATA4(0xf5, 0x87, 0x51, 0x65),
CRYPTO_MPI_LIMB_DATA4(0xe4, 0x2c, 0x9b, 0x62),
CRYPTO_MPI_LIMB_DATA4(0x3e, 0xe0, 0x21, 0x5b),
CRYPTO_MPI_LIMB_DATA4(0x09, 0x41, 0xe5, 0xb5),
CRYPTO_MPI_LIMB_DATA4(0x92, 0x8c, 0x7c, 0x1b),
CRYPTO_MPI_LIMB_DATA4(0x8a, 0x3e, 0x10, 0x2a),

```

```

CRYPTO_MPI_LIMB_DATA4(0x81, 0xd5, 0xd4, 0x03),
CRYPTO_MPI_LIMB_DATA4(0x62, 0x3b, 0x99, 0x4b),
CRYPTO_MPI_LIMB_DATA4(0x6c, 0xd1, 0x3d, 0x74),
CRYPTO_MPI_LIMB_DATA4(0x0f, 0x3a, 0x9d, 0xd2),
CRYPTO_MPI_LIMB_DATA4(0x2a, 0x67, 0xbe, 0x47),
CRYPTO_MPI_LIMB_DATA4(0x3b, 0x80, 0x4d, 0xe7),
CRYPTO_MPI_LIMB_DATA4(0xae, 0x5a, 0x8f, 0xb9),
CRYPTO_MPI_LIMB_DATA4(0xdd, 0x6f, 0x3c, 0x98),
CRYPTO_MPI_LIMB_DATA4(0xea, 0x38, 0x6c, 0x50),
CRYPTO_MPI_LIMB_DATA4(0x27, 0x5f, 0xea, 0x19),
CRYPTO_MPI_LIMB_DATA4(0x7d, 0xd8, 0x9f, 0x00),
CRYPTO_MPI_LIMB_DATA4(0x78, 0xc5, 0x05, 0x72),
CRYPTO_MPI_LIMB_DATA4(0x4c, 0x5a, 0x13, 0xaf),
CRYPTO_MPI_LIMB_DATA4(0xbf, 0x64, 0x79, 0x69),
CRYPTO_MPI_LIMB_DATA4(0xbd, 0x75, 0x57, 0xae),
CRYPTO_MPI_LIMB_DATA4(0x4f, 0xd6, 0xce, 0xe9),
CRYPTO_MPI_LIMB_DATA4(0xd9, 0x81, 0x48, 0x2f),
CRYPTO_MPI_LIMB_DATA4(0xef, 0x36, 0x86, 0x0c),
CRYPTO_MPI_LIMB_DATA4(0xd5, 0x5e, 0x29, 0xd4),
CRYPTO_MPI_LIMB_DATA4(0xb1, 0xa3, 0xee, 0xe0),
CRYPTO_MPI_LIMB_DATA4(0xba, 0xe0, 0x38, 0xd9),
CRYPTO_MPI_LIMB_DATA4(0xc6, 0x01, 0xcb, 0xff),
CRYPTO_MPI_LIMB_DATA4(0xdb, 0x56, 0x09, 0x50),
CRYPTO_MPI_LIMB_DATA4(0xce, 0x10, 0x3c, 0x23),
CRYPTO_MPI_LIMB_DATA4(0x99, 0x6c, 0x7f, 0x39),
CRYPTO_MPI_LIMB_DATA4(0xd4, 0x86, 0xfa, 0xa0)
};

static const CRYPTO_MPI _RSA2048_N = {
    CRYPTO_MPI_INIT_RO(_RSA2048_N_aLimbs)
};

__MPI_LITERAL_BEGIN(static, _RSA2048_E)
    __MPI_LITERAL_DATA(0x01, 0x00, 0x01, 0x00)
__MPI_LITERAL_END(static, _RSA2048_E, 17)

__MPI_LITERAL_BEGIN(static, _RSA2048_P)
    __MPI_LITERAL_DATA(0x29, 0x78, 0x79, 0xe2),
    __MPI_LITERAL_DATA(0x5c, 0x0d, 0xeb, 0xed),
    __MPI_LITERAL_DATA(0x2a, 0x79, 0xaf, 0x00),
    __MPI_LITERAL_DATA(0xe0, 0xe7, 0xc9, 0x58),
    __MPI_LITERAL_DATA(0x11, 0x17, 0x42, 0x0d),
    __MPI_LITERAL_DATA(0xa5, 0x11, 0x99, 0x1b),
    __MPI_LITERAL_DATA(0x26, 0x62, 0x37, 0x3c),
    __MPI_LITERAL_DATA(0xbf, 0xdc, 0xa4, 0x78),
    __MPI_LITERAL_DATA(0x3e, 0x95, 0xd6, 0x8d),
    __MPI_LITERAL_DATA(0x8c, 0x92, 0xde, 0x78),
    __MPI_LITERAL_DATA(0x7a, 0x03, 0xd9, 0xd2),
    __MPI_LITERAL_DATA(0x2d, 0xb1, 0x35, 0x4c),
    __MPI_LITERAL_DATA(0x9e, 0x4b, 0x71, 0x29),
    __MPI_LITERAL_DATA(0xf8, 0x8d, 0x7e, 0x56),
    __MPI_LITERAL_DATA(0x33, 0x42, 0xd7, 0xd7),
    __MPI_LITERAL_DATA(0x1a, 0xe5, 0xcc, 0xb7),
    __MPI_LITERAL_DATA(0x14, 0x78, 0x8d, 0x29),
    __MPI_LITERAL_DATA(0x5d, 0x19, 0xde, 0x8c),
    __MPI_LITERAL_DATA(0x14, 0xd6, 0x51, 0xc5),
    __MPI_LITERAL_DATA(0x34, 0xe7, 0xfe, 0x5b),
    __MPI_LITERAL_DATA(0x37, 0xb8, 0xf4, 0x3f),
    __MPI_LITERAL_DATA(0x29, 0x8d, 0x38, 0xa0),
    __MPI_LITERAL_DATA(0x41, 0xb8, 0xd9, 0x82),
    __MPI_LITERAL_DATA(0x05, 0xf5, 0xd2, 0xf7),
    __MPI_LITERAL_DATA(0x7e, 0x23, 0xf4, 0x46),
    __MPI_LITERAL_DATA(0xde, 0x69, 0x11, 0x45),
    __MPI_LITERAL_DATA(0x22, 0x33, 0x6a, 0xdf),
    __MPI_LITERAL_DATA(0x38, 0x3d, 0xff, 0x14),
    __MPI_LITERAL_DATA(0xaa, 0xd5, 0xb7, 0x17),
    __MPI_LITERAL_DATA(0x4f, 0xc2, 0x40, 0x0f),
    __MPI_LITERAL_DATA(0x67, 0x80, 0x53, 0x55),
    __MPI_LITERAL_DATA(0xbd, 0x37, 0xc2, 0xc6)
__MPI_LITERAL_END(static, _RSA2048_P, 1024)

__MPI_LITERAL_BEGIN(static, _RSA2048_Q)
    __MPI_LITERAL_DATA(0x8f, 0xe0, 0xab, 0xab),
    __MPI_LITERAL_DATA(0x0c, 0xe5, 0xdb, 0x1b),
    __MPI_LITERAL_DATA(0xb8, 0x29, 0x3f, 0x90),
    __MPI_LITERAL_DATA(0x4f, 0x91, 0xee, 0x24),

```

```
__MPI_LITERAL_DATA(0xae, 0xc2, 0x70, 0x7d),
__MPI_LITERAL_DATA(0x3e, 0x0e, 0xd0, 0x3f),
__MPI_LITERAL_DATA(0x23, 0x8b, 0x16, 0xef),
__MPI_LITERAL_DATA(0x5e, 0x1e, 0xb8, 0x1d),
__MPI_LITERAL_DATA(0x59, 0x6f, 0xdd, 0x12),
__MPI_LITERAL_DATA(0x62, 0xfb, 0xe8, 0xa0),
__MPI_LITERAL_DATA(0x04, 0xca, 0xd3, 0x2e),
__MPI_LITERAL_DATA(0x20, 0xf4, 0x5b, 0xb0),
__MPI_LITERAL_DATA(0xb9, 0x4f, 0xa6, 0x32),
__MPI_LITERAL_DATA(0x5d, 0xef, 0x4f, 0x87),
__MPI_LITERAL_DATA(0x2b, 0x52, 0x87, 0x20),
__MPI_LITERAL_DATA(0x34, 0x2f, 0xed, 0x6d),
__MPI_LITERAL_DATA(0x4b, 0x02, 0x61, 0x46),
__MPI_LITERAL_DATA(0x73, 0x76, 0x1b, 0x44),
__MPI_LITERAL_DATA(0xd4, 0x5b, 0x64, 0xf7),
__MPI_LITERAL_DATA(0xff, 0xf7, 0xb3, 0xe6),
__MPI_LITERAL_DATA(0xce, 0x08, 0x10, 0x8f),
__MPI_LITERAL_DATA(0xd9, 0x0d, 0x60, 0xbe),
__MPI_LITERAL_DATA(0xef, 0x62, 0x81, 0x67),
__MPI_LITERAL_DATA(0xa4, 0x5e, 0x0c, 0xdb),
__MPI_LITERAL_DATA(0x5b, 0x72, 0x8f, 0x8b),
__MPI_LITERAL_DATA(0xe3, 0xf0, 0x5a, 0x73),
__MPI_LITERAL_DATA(0x5f, 0xb4, 0xba, 0xa2),
__MPI_LITERAL_DATA(0xd8, 0x24, 0x6e, 0x34),
__MPI_LITERAL_DATA(0xce, 0xe0, 0x95, 0x61),
__MPI_LITERAL_DATA(0xee, 0xb4, 0xd0, 0x5e),
__MPI_LITERAL_DATA(0x2a, 0x02, 0x7b, 0x79),
__MPI_LITERAL_DATA(0x13, 0xeb, 0x56, 0xcf)
__MPI_LITERAL_END(static, _RSA2048_Q, 1024)

__MPI_LITERAL_BEGIN(static, _RSA2048_DP)
__MPI_LITERAL_DATA(0x79, 0x73, 0x0c, 0xf7),
__MPI_LITERAL_DATA(0xaa, 0x65, 0xbe, 0x0c),
__MPI_LITERAL_DATA(0x84, 0x0c, 0x7f, 0x6e),
__MPI_LITERAL_DATA(0x8a, 0x0d, 0x13, 0x59),
__MPI_LITERAL_DATA(0x5f, 0x79, 0x04, 0xc8),
__MPI_LITERAL_DATA(0x42, 0x82, 0x03, 0x72),
__MPI_LITERAL_DATA(0x45, 0x5c, 0x7f, 0x22),
__MPI_LITERAL_DATA(0x10, 0xe6, 0x0d, 0x9c),
__MPI_LITERAL_DATA(0x71, 0x10, 0x07, 0xf4),
__MPI_LITERAL_DATA(0x5f, 0xff, 0x91, 0x33),
__MPI_LITERAL_DATA(0x44, 0x14, 0x3e, 0x95),
__MPI_LITERAL_DATA(0x67, 0xe9, 0x18, 0xc1),
__MPI_LITERAL_DATA(0xd0, 0xe7, 0xd6, 0x8d),
__MPI_LITERAL_DATA(0xfa, 0xa5, 0x16, 0xaf),
__MPI_LITERAL_DATA(0x20, 0xb3, 0x4f, 0x57),
__MPI_LITERAL_DATA(0x7d, 0xda, 0x1e, 0x95),
__MPI_LITERAL_DATA(0x19, 0x47, 0x1c, 0x1e),
__MPI_LITERAL_DATA(0x55, 0x0d, 0xc4, 0x98),
__MPI_LITERAL_DATA(0xa5, 0x83, 0xdd, 0x5c),
__MPI_LITERAL_DATA(0xdb, 0x30, 0x5a, 0xba),
__MPI_LITERAL_DATA(0xb7, 0xb4, 0x60, 0x43),
__MPI_LITERAL_DATA(0x6c, 0x8f, 0x16, 0x4f),
__MPI_LITERAL_DATA(0xdc, 0x4b, 0x51, 0xda),
__MPI_LITERAL_DATA(0xc5, 0xb4, 0xb9, 0x1f),
__MPI_LITERAL_DATA(0xf9, 0x3b, 0xb9, 0x97),
__MPI_LITERAL_DATA(0xc7, 0x20, 0xef, 0x85),
__MPI_LITERAL_DATA(0xbb, 0x4c, 0xff, 0x46),
__MPI_LITERAL_DATA(0xf5, 0xff, 0xf4, 0x29),
__MPI_LITERAL_DATA(0x68, 0xf2, 0x4c, 0xf4),
__MPI_LITERAL_DATA(0x01, 0x9d, 0x8b, 0x9d),
__MPI_LITERAL_DATA(0xf4, 0xd5, 0xd3, 0xba),
__MPI_LITERAL_DATA(0x0d, 0xda, 0x79, 0x77)
__MPI_LITERAL_END(static, _RSA2048_DP, 1023)

__MPI_LITERAL_BEGIN(static, _RSA2048_DQ)
__MPI_LITERAL_DATA(0x9d, 0x5c, 0x6a, 0x09),
__MPI_LITERAL_DATA(0xe9, 0xf6, 0x40, 0x1d),
__MPI_LITERAL_DATA(0x18, 0x8f, 0x7c, 0x4d),
__MPI_LITERAL_DATA(0x5f, 0x3d, 0xe5, 0x78),
__MPI_LITERAL_DATA(0x6d, 0xbe, 0xb0, 0xa4),
__MPI_LITERAL_DATA(0x6b, 0x70, 0xc8, 0x48),
__MPI_LITERAL_DATA(0x3b, 0x5b, 0xee, 0x16),
__MPI_LITERAL_DATA(0xf0, 0xd2, 0x64, 0xc2),
__MPI_LITERAL_DATA(0x30, 0xc2, 0x64, 0x9a),
__MPI_LITERAL_DATA(0x42, 0x84, 0x00, 0xfa),
```

```

__MPI_LITERAL_DATA(0x0b, 0xea, 0x77, 0xe3),
__MPI_LITERAL_DATA(0x1e, 0x9f, 0xf2, 0xc3),
__MPI_LITERAL_DATA(0xd0, 0x52, 0x34, 0xb3),
__MPI_LITERAL_DATA(0x9b, 0x6a, 0x80, 0xb1),
__MPI_LITERAL_DATA(0x93, 0x7e, 0x68, 0xc0),
__MPI_LITERAL_DATA(0xbc, 0xc7, 0xd9, 0x35),
__MPI_LITERAL_DATA(0x89, 0xd8, 0x5e, 0x31),
__MPI_LITERAL_DATA(0xd7, 0x5f, 0x82, 0xe9),
__MPI_LITERAL_DATA(0xd1, 0xad, 0xec, 0x4d),
__MPI_LITERAL_DATA(0xb4, 0x9e, 0x91, 0x28),
__MPI_LITERAL_DATA(0xe4, 0xee, 0xae, 0x36),
__MPI_LITERAL_DATA(0x4a, 0x57, 0xe2, 0x42),
__MPI_LITERAL_DATA(0x4a, 0xe5, 0xb6, 0x54),
__MPI_LITERAL_DATA(0x1a, 0x4a, 0x0e, 0x04),
__MPI_LITERAL_DATA(0x31, 0x59, 0x76, 0xaa),
__MPI_LITERAL_DATA(0xb7, 0x52, 0xd3, 0xf0),
__MPI_LITERAL_DATA(0xd7, 0xd8, 0xbf, 0x09),
__MPI_LITERAL_DATA(0xab, 0x15, 0x1f, 0x75),
__MPI_LITERAL_DATA(0x85, 0x0c, 0xef, 0xa7),
__MPI_LITERAL_DATA(0xa4, 0x1b, 0xd7, 0xdc),
__MPI_LITERAL_DATA(0x1e, 0x3f, 0x70, 0x1f),
__MPI_LITERAL_DATA(0x09, 0x17, 0x33, 0x51)
__MPI_LITERAL_END(static, _RSA2048_DQ, 1023)

__MPI_LITERAL_BEGIN(static, _RSA2048_QINV)
__MPI_LITERAL_DATA(0x0d, 0xa3, 0x5a, 0xe9),
__MPI_LITERAL_DATA(0x0f, 0x46, 0x32, 0x97),
__MPI_LITERAL_DATA(0x59, 0xf1, 0xd0, 0xec),
__MPI_LITERAL_DATA(0x08, 0xd6, 0x56, 0xd0),
__MPI_LITERAL_DATA(0xc9, 0x0e, 0x2a, 0x04),
__MPI_LITERAL_DATA(0x66, 0x0c, 0xa6, 0xbff),
__MPI_LITERAL_DATA(0x5b, 0xcb, 0x3b, 0x24),
__MPI_LITERAL_DATA(0x76, 0xab, 0x72, 0xb8),
__MPI_LITERAL_DATA(0x65, 0x89, 0x79, 0x87),
__MPI_LITERAL_DATA(0xa8, 0xc7, 0x51, 0xd6),
__MPI_LITERAL_DATA(0x8a, 0x50, 0x54, 0xda),
__MPI_LITERAL_DATA(0x07, 0x93, 0x97, 0xa5),
__MPI_LITERAL_DATA(0x3b, 0x84, 0x56, 0xa7),
__MPI_LITERAL_DATA(0x4b, 0x5a, 0xab, 0x2d),
__MPI_LITERAL_DATA(0x79, 0xa7, 0xbf, 0x1a),
__MPI_LITERAL_DATA(0xc5, 0xf6, 0x29, 0x70),
__MPI_LITERAL_DATA(0xc3, 0xcd, 0x3a, 0xde),
__MPI_LITERAL_DATA(0x58, 0x7b, 0x40, 0x52),
__MPI_LITERAL_DATA(0xd2, 0x10, 0x41, 0x73),
__MPI_LITERAL_DATA(0xc1, 0x11, 0x02, 0xdf),
__MPI_LITERAL_DATA(0xd1, 0xba, 0x9d, 0x11),
__MPI_LITERAL_DATA(0xeb, 0x88, 0xcb, 0xaf),
__MPI_LITERAL_DATA(0x19, 0x7d, 0x96, 0xac),
__MPI_LITERAL_DATA(0xeb, 0x3f, 0xc0, 0x58),
__MPI_LITERAL_DATA(0x92, 0x95, 0xf2, 0xe6),
__MPI_LITERAL_DATA(0x5d, 0x00, 0x42, 0x59),
__MPI_LITERAL_DATA(0xd8, 0xc4, 0x00, 0xab),
__MPI_LITERAL_DATA(0xde, 0x34, 0x3d, 0x4f),
__MPI_LITERAL_DATA(0xc6, 0xcc, 0xb8, 0xd6),
__MPI_LITERAL_DATA(0xbe, 0x74, 0xef, 0x6d),
__MPI_LITERAL_DATA(0xcb, 0x98, 0xba, 0x9d),
__MPI_LITERAL_DATA(0x00, 0xcd, 0x72, 0x08)
__MPI_LITERAL_END(static, _RSA2048_QINV, 1020)

*****Benchmark parameterization.*****
static const BENCH_ALG _aBenchAlgs[] = {
    { "Basic, fast", CRYPTO_MPI_ModExp_Basic_Fast },
    { "Basic, ladder", CRYPTO_MPI_ModExp_Basic_Ladder },
    { "Basic, 2b, FW", CRYPTO_MPI_ModExp_Basic_2b_FW },
    { "Basic, 3b, FW", CRYPTO_MPI_ModExp_Basic_3b_FW },
    { "Basic, 4b, FW", CRYPTO_MPI_ModExp_Basic_4b_FW },
    { "Basic, 5b, FW", CRYPTO_MPI_ModExp_Basic_5b_FW },
    { "Basic, 6b, FW", CRYPTO_MPI_ModExp_Basic_6b_FW },
    { NULL, NULL },
    { "Basic, 2b, RM", CRYPTO_MPI_ModExp_Basic_2b_RM },
    { "Basic, 3b, RM", CRYPTO_MPI_ModExp_Basic_3b_RM },
    { "Basic, 4b, RM", CRYPTO_MPI_ModExp_Basic_4b_RM }
};

```

```

    {
        "Basic, 5b, RM",
        CRYPTO_MPI_ModExp_Basic_5b_RM
    },
    {
        "Basic, 6b, RM",
        CRYPTO_MPI_ModExp_Basic_6b_RM
    },
    NULL,
    NULL
},
{
    "Barrett, fast",
    CRYPTO_MPI_ModExp_Barrett_Fast
},
{
    "Barrett, ladder",
    CRYPTO_MPI_ModExp_Barrett_Ladder
},
{
    "Barrett, 2b, FW",
    CRYPTO_MPI_ModExp_Barrett_2b_FW
},
{
    "Barrett, 3b, FW",
    CRYPTO_MPI_ModExp_Barrett_3b_FW
},
{
    "Barrett, 4b, FW",
    CRYPTO_MPI_ModExp_Barrett_4b_FW
},
{
    "Barrett, 5b, FW",
    CRYPTO_MPI_ModExp_Barrett_5b_FW
},
{
    "Barrett, 6b, FW",
    CRYPTO_MPI_ModExp_Barrett_6b_FW
},
NULL,
NULL
},
{
    "Barrett, 2b, RM",
    CRYPTO_MPI_ModExp_Barrett_2b_RM
},
{
    "Barrett, 3b, RM",
    CRYPTO_MPI_ModExp_Barrett_3b_RM
},
{
    "Barrett, 4b, RM",
    CRYPTO_MPI_ModExp_Barrett_4b_RM
},
{
    "Barrett, 5b, RM",
    CRYPTO_MPI_ModExp_Barrett_5b_RM
},
{
    "Barrett, 6b, RM",
    CRYPTO_MPI_ModExp_Barrett_6b_RM
},
NULL,
NULL
},
{
    "Montgomery, fast",
    CRYPTO_MPI_ModExp_Montgomery_Fast
},
{
    "Montgomery, ladder",
    CRYPTO_MPI_ModExp_Montgomery_Ladder
},
{
    "Montgomery, 2b, FW",
    CRYPTO_MPI_ModExp_Montgomery_2b_FW
},
{
    "Montgomery, 2b, FW, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_2bFW_EFM32_CRYPTO }, )
},
{
    "Montgomery, 3b, FW",
    CRYPTO_MPI_ModExp_Montgomery_3b_FW
},
{
    "Montgomery, 3b, FW, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_3bFW_EFM32_CRYPTO }, )
},
{
    "Montgomery, 4b, FW",
    CRYPTO_MPI_ModExp_Montgomery_4b_FW
},
{
    "Montgomery, 4b, FW, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_4bFW_EFM32_CRYPTO }, )
},
{
    "Montgomery, 5b, FW",
    CRYPTO_MPI_ModExp_Montgomery_5b_FW
},
{
    "Montgomery, 5b, FW, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_5bFW_EFM32_CRYPTO }, )
},
{
    "Montgomery, 6b, FW",
    CRYPTO_MPI_ModExp_Montgomery_6b_FW
},
{
    "Montgomery, 6b, FW, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_6bFW_EFM32_CRYPTO }, )
},
NULL,
NULL
},
{
    "Montgomery, 2b, RM",
    CRYPTO_MPI_ModExp_Montgomery_2b_RM
},
{
    "Montgomery, 2b, RM, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_2bRM_EFM32_CRYPTO }, )
},
{
    "Montgomery, 3b, RM",
    CRYPTO_MPI_ModExp_Montgomery_3b_RM
},
{
    "Montgomery, 3b, RM, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_3bRM_EFM32_CRYPTO }, )
},
{
    "Montgomery, 4b, RM",
    CRYPTO_MPI_ModExp_Montgomery_4b_RM
},
{
    "Montgomery, 4b, RM, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_4bRM_EFM32_CRYPTO }, )
},
{
    "Montgomery, 5b, RM",
    CRYPTO_MPI_ModExp_Montgomery_5b_RM
},
{
    "Montgomery, 5b, RM, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_5bRM_EFM32_CRYPTO }, )
},
{
    "Montgomery, 6b, RM",
    CRYPTO_MPI_ModExp_Montgomery_6b_RM
},
{
    "Montgomery, 6b, RM, EFM32",
    CRYPTO_MPI_ModExp_Montgomery_6bRM_EFM32_CRYPTO }, )
},
NULL,
NULL
},
{
    "Configured",
    CRYPTO_MPI_ModExp
};

static const BENCH_SCENARIO _aBenchScenarios[] = {
    { "CRT private key, exponent length = modulus length",      _BenchmarkModExp_Private_CRT
    },
#ifndef INCLUDE_PLAIN_PRIVATE
    { "Non-CRT private key, exponent length = modulus
length",      _BenchmarkModExp_Private_Plain },
#endif
    { "Public key, exponent length = 17 bits",                  _BenchmarkModExp_Public
    }
};

static const BENCH_KEY _aBenchKeys[] = {
#ifndef INCLUDE_SMALL_MODULI
    { &_RSA128_N,   &_RSA128_E,   &_RSA128_P,   &_RSA128_Q,   &_RSA128_DP,   &_RSA128_DQ,
    &_RSA128_QINV },
    { &_RSA256_N,   &_RSA256_E,   &_RSA256_P,   &_RSA256_Q,   &_RSA256_DP,   &_RSA256_DQ,
    &_RSA256_QINV },
    { &_RSA512_N,   &_RSA512_E,   &_RSA512_P,   &_RSA512_Q,   &_RSA512_DP,   &_RSA512_DQ,
    &_RSA512_QINV },
    { &_RSA1024_N,  &_RSA1024_E,  &_RSA1024_P,  &_RSA1024_Q,  &_RSA1024_DP,  &_RSA1024_DQ,  &_RSA1024_QINV },
    { &_RSA2048_N,  &_RSA2048_E,  &_RSA2048_P,  &_RSA2048_Q,  &_RSA2048_DP,  &_RSA2048_DQ,  &_RSA2048_QINV }
}
};

/*****************
*      Static data
*
***** */
static MPI_UNIT          _aUnits[MAX_CHUNKS];

```

```

static SEGGER_MEM_CONTEXT      _MemContext;
static SEGGER_MEM_SELFTEST_HEAP _Heap;
static unsigned                _AlgIndex;
static unsigned                _KeyIndex;
static float                  _aBaselinePerf[SEGGER_COUNTOF(_aBenchKeys)];
static float                  _aBaselineMem [SEGGER_COUNTOF(_aBenchKeys)];

//*****************************************************************************
/*
*      Static code
*/
//*****************************************************************************

/*_ConvertTicksToSeconds( )

* Function description
*   Convert ticks to seconds.

* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static float _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0f;
}

//*****************************************************************************
/*
*      _BenchmarkSingleModExp( )

* Function description
*   Count the number of encryptions completed in one second using
*   the public key parameters N, E.
*
* Parameters
*   pfModExp  - Pointer to modular exponentiation implementation.
*   pN        - Pointer to MPI containing modulus.
*   pExponent - Pointer to MPI containing exponent.
*/
static void _BenchmarkSingleModExp(MODEXP_FUNC pfModExp, const CRYPTO_MPI *pN, const CRYPTO_MPI *pExponent)
{
    CRYPTO_MPI Data;
    U64 OneSecond;
    U64 T0;
    U64 Elapsed;
    int Loops;
    int Status;
    unsigned PeakBytes;
    unsigned ChunkSize;
    float PerfMultiplier;
    float MemMultiplier;
    float Time;
    //
    PeakBytes = 0;
    Loops = 0;
    //
    ChunkSize = CRYPTO_MPI_BYTES_REQUIRED(2*CRYPTO_MPI_BitCount(pN)+CRYPTO_MPI_BYTES_PER_LIMB-1) + 2*CRYPTO_MP
    CRYPTO_MPI_SetChunkSize(ChunkSize);
    //
    // Create fixed plaintext.
    //
    CRYPTO_MPI_Init(&Data, &_MemContext);
    CRYPTO_MPI_LoadHex(&Data, "123456789ABCDEF123456789ABCDEF0123456789ABCDEF", 0);
    CRYPTO_MPI_Mod(&Data, pN, &_MemContext);
    //
    // Count number of modular exponentiations completed in 1s.
    //
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        Status = pfModExp(&Data, pExponent, pN, &_MemContext);

```

```

PeakBytes = SEGGER_MAX(PeakBytes, _Heap.Stats.NumInUseMax * ChunkSize);
++Loops;
Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
} while (Status >= 0 && Elapsed < OneSecond);
//
CRYPTO_MPI_Kill(&Data);
//
Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
if (_AlgIndex == 0) {
    _aBaselinePerf[_KeyIndex] = Time;
    _aBaselineMem [_KeyIndex] = (float)PeakBytes;
}
PerfMultiplier = _aBaselinePerf[_KeyIndex] / (float)Time;
MemMultiplier = (float)PeakBytes / _aBaselineMem [_KeyIndex];
if (Status < 0) {
    SEGGER_SYS_IO_Printf(" -Fail- -Fail- | ");
} else {
    SEGGER_SYS_IO_Printf("%8.2f %5.2fx %8u %5.2fx
| ", Time, PerfMultiplier, PeakBytes, MemMultiplier);
}
}

/*********************************************
*
*      _BenchmarkModExp_Public()
*
*  Function description
*      Benchmark public key operation.
*
*  Parameters
*      pfModExp - Modular exponentiation function to benchmark.
*      pKey     - Pointer to key ti benchmark with.
*/
static void _BenchmarkModExp_Public(MODEXP_FUNC pfModExp, const BENCH_KEY *pKey) {
    _BenchmarkSingleModExp(pfModExp, pKey->pN, pKey->pE);
}

#if INCLUDE_PLAIN_PRIVATE

/*********************************************
*
*      _BenchmarkModExp_Private_Plain()
*
*  Function description
*      Benchmark non-CRT private key operation.
*
*  Parameters
*      pfModExp - Modular exponentiation function to benchmark.
*      pKey     - Pointer to key ti benchmark with.
*/
static void _BenchmarkModExp_Private_Plain(MODEXP_FUNC pfModExp, const BENCH_KEY *pKey) {
    CRYPTO_MPI D;
    CRYPTO_MPI P;
    CRYPTO_MPI Q;
    CRYPTO_MPI x;
    //
    // Compute non-CRT form of decryption exponent, d = modinv(e, lcm(p-1, q-1))
    //
    CRYPTO_MPI_Init(&x, &_MemContext);
    CRYPTO_MPI_Init(&D, &_MemContext);
    CRYPTO_MPI_Init(&P, &_MemContext);
    CRYPTO_MPI_Init(&Q, &_MemContext);
    CRYPTO_MPI_Assign(&P, pKey->pP);
    CRYPTO_MPI_Assign(&Q, pKey->pQ);
    CRYPTO_MPI_Dec(&P);
    CRYPTO_MPI_Dec(&Q);
    CRYPTO_MPI_LCM(&x, &P, &Q, &_MemContext);           // x = lcm(p-1, q-1)
    CRYPTO_MPI_ModInvEx(&D, pKey->pE, &x, &_MemContext); // d = modinv(e, lcm(p-1, q-1))
    CRYPTO_MPI_Kill(&P);
    CRYPTO_MPI_Kill(&Q);
    CRYPTO_MPI_Kill(&x);
    //
    _BenchmarkSingleModExp(pfModExp, pKey->pN, &D);
    //
    CRYPTO_MPI_Kill(&D);
}

```

```
#endif

//****************************************************************************
*
*      _BenchmarkModExp_Private_CRT()
*
*  Function description
*      Benchmark private key operation in CRT form.
*
*  Parameters
*      pfModExp - Modular exponentiation function to benchmark.
*      pKey     - Pointer to key to benchmark with.
*/
static void _BenchmarkModExp_Private_CRT(MODEXP_FUNC pfModExp, const BENCH_KEY *pKey) {
    CRYPTO_MPI Data;
    CRYPTO_MPI a;
    CRYPTO_MPI b;
    U64        OneSecond;
    U64        T0;
    U64        Elapsed;
    int        Loops;
    int        Status;
    unsigned   ChunkSize;
    unsigned   PeakBytes;
    float      Time;
    float      PerfMultiplier;
    float      MemMultiplier;
    //
    ChunkSize = CRYPTO_MPI_BYTES_REQUIRED(2*CRYPTO_MPI_BitCount(pKey-
>pP)+CRYPTO_MPI_BYTES_PER_LIMB-1) + 2*CRYPTO_MPI_BYTES_PER_LIMB;
    CRYPTO_MPI_SetChunkSize(ChunkSize);
    //
    // Make PC-lint quiet, it's dataflow analysis provides false positives.
    //
    Loops      = 0;
    Elapsed    = 0;
    PeakBytes  = 0;
    //
    // Create fixed plaintext.
    //
    CRYPTO_MPI_Init(&Data, &_MemContext);
    CRYPTO_MPI_Init(&a, &_MemContext);
    CRYPTO_MPI_Init(&b, &_MemContext);
    //
    CRYPTO_CHECK(CRYPTO_MPI_LoadHex(&Data, "123456789ABCDEF123456789ABCDEF0123456789ABCDEF", 0));
    CRYPTO_CHECK(CRYPTO_MPI_Mod    (&Data, pKey->pN, &_MemContext));
    //
    // Count number of modular exponentiations completed in 1s.
    //
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    T0 = SEGGER_SYS_OS_GetTimer();
    do {
        //
        // Apply Chinese Remainder Theorem. We assume that ciphertext
        // is already reduced modulo p.
        //
        _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
        //
        CRYPTO_CHECK(CRYPTO_MPI_Assign(&a, &Data));
        CRYPTO_CHECK(pfModExp(&a, pKey->pDP, pKey->pP, &_MemContext)); // a = cipher^dp
        (mod p)
        CRYPTO_CHECK(CRYPTO_MPI_Assign(&b, &Data));
        CRYPTO_CHECK(pfModExp(&b, pKey->pDQ, pKey->pQ, &_MemContext)); // b = cipher^dq
        (mod q)
        //
        // plaintext = b + q * (((a-b)*u) % p)
        //
        CRYPTO_CHECK(CRYPTO_MPI_Move(&Data, &a)); // a
        CRYPTO_CHECK(CRYPTO_MPI_Sub(&Data, &b));
        // a-b could be negative as p < q.
        while (CRYPTO_MPI_IsNegative(&Data)) {
            CRYPTO_CHECK(CRYPTO_MPI_Add(&Data, pKey->pP));
        }
        CRYPTO_CHECK(CRYPTO_MPI_ModMul(&Data, pKey->pQInv, pKey->pP, &_MemContext)); // (a-b)*qinv mod p
    }
}
```

```

CRYPTO_CHECK(CRYPTO_MPI_Mul(&Data, pKey->pQ, &_MemContext));           // q *
((a-b)*qinv mod p)
CRYPTO_CHECK(CRYPTO_MPI_Add(&Data, &b));                                // b + q *
((a-b)*qinv mod p)
//
PeakBytes = SEGGER_MAX(PeakBytes, _Heap.Stats.NumInUseMax * ChunkSize);
//
++Loops;
Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
} while (Status >= 0 && Elapsed < OneSecond);
//
Finally:
CRYPTO_MPI_Kill(&Data);
CRYPTO_MPI_Kill(&a);
CRYPTO_MPI_Kill(&b);
if (Status < 0 || Loops == 0) {
    SEGGER_SYS_IO_Printf(" -Fail- -Fail- | ");
} else {
    Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
    if (_AlgIndex == 0) {
        _aBaselinePerf[_KeyIndex] = Time;
        _aBaselineMem[_KeyIndex] = (float)PeakBytes;
    }
    PerfMultiplier = _aBaselinePerf[_KeyIndex] / (float)Time;
    MemMultiplier = (float)PeakBytes / _aBaselineMem[_KeyIndex];
    if (Status < 0) {
        SEGGER_SYS_IO_Printf(" -Fail- -Fail- | ");
    } else {
        SEGGER_SYS_IO_Printf("%8.2f %5.2fx %8u %5.2fx
| ", Time, PerfMultiplier, PeakBytes, MemMultiplier);
    }
}
}

/*********************************************
*
*      _PrintSeparator()
*
*  Function description
*      Print row separator for table.
*/
static void _PrintSeparator(void) {
unsigned i;
//
SEGGER_SYS_IO_Printf("+-----+");
for (i = 0; i < SEGGER_COUNTOF(_aBenchKeys); ++i) {
    SEGGER_SYS_IO_Printf("------+");
}
SEGGER_SYS_IO_Printf("\n");
}

/*********************************************
*
*      _PrintHeader()
*
*  Function description
*      Print column headers for table.
*/
static void _PrintHeader(void) {
unsigned i;
/
_PrintSeparator();
SEGGER_SYS_IO_Printf(" | Modulus | ");
for (i = 0; i < SEGGER_COUNTOF(_aBenchKeys); ++i) {
    SEGGER_SYS_IO_Printf(" %25d bits | ", CRYPTO_MPI_BitCount(_aBenchKeys[i].pN));
}
SEGGER_SYS_IO_Printf("\n");
SEGGER_SYS_IO_Printf(" | Algorithm | ");
for (i = 0; i < SEGGER_COUNTOF(_aBenchKeys); ++i) {
    SEGGER_SYS_IO_Printf(" %7s %6s %7s %6s | ", "Time", "x", "Memory", "x");
}
SEGGER_SYS_IO_Printf("\n");
_PrintSeparator();
}

/*********************************************

```

```

/*
*      Public code
*
***** End of file *****/
*/



/*
*      MainTask()
*
*      Function description
*      Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    unsigned i;
    // CRYPTO_Init();
    SEGGER_SYS_Init();
    SEGGER_MEM_SELFTEST_HEAP_Init(&_MemContext, &_Heap, _aUnits, MAX_CHUNKS, sizeof(MPI_UNIT));
    //
    SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
    SEGGER_SYS_IO_Printf("Modular Exponentiation Benchmark compiled " __DATE__ " "
" __TIME__ "\n\n");
    //
    SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
    if (SEGGER_SYS_GetProcessorSpeed() > 0) {
        SEGGER_SYS_IO_Printf("System: Processor speed      = %.3f MHz
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
    }
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_VERSION      = %u
[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
    SEGGER_SYS_IO_Printf("Config:   CRYPTO_MPI_BITS_PER_LIMB = %u\n",
CRYPTO_MPI_BITS_PER_LIMB);
    SEGGER_SYS_IO_Printf("\n");
    //
    SEGGER_SYS_IO_Printf("Modular Arithmetic Performance\n");
    SEGGER_SYS_IO_Printf("=====*\n\n");
    //
    for (i = 0; i < SEGGER_COUNTOF(_aBenchScenarios); ++i) {
        SEGGER_SYS_IO_Printf("%s, all times in ms\n\n", _aBenchScenarios[i].pText);
        _PrintHeader();
        for (_AlgIndex = 0; _AlgIndex < SEGGER_COUNTOF(_aBenchAlgs); ++_AlgIndex) {
            if (_aBenchAlgs[_AlgIndex].pfModExp == 0) {
                _PrintSeparator();
            } else {
                SEGGER_SYS_IO_Printf(" | %-25s | ", _aBenchAlgs[_AlgIndex].pText);
                for (_KeyIndex = 0; _KeyIndex < SEGGER_COUNTOF(_aBenchKeys); ++_KeyIndex) {

_aBenchScenarios[i].pfBenchFunc(_aBenchAlgs[_AlgIndex].pfModExp, &_aBenchKeys[_KeyIndex]);
                }
                SEGGER_SYS_IO_Printf("\n");
            }
        }
        _PrintSeparator();
        SEGGER_SYS_IO_Printf("\n");
    }
    //
    SEGGER_SYS_IO_Printf("Benchmark complete\n");
    SEGGER_SYS_OS_PauseBeforeHalt();
    SEGGER_SYS_OS_Halt(0);
}

***** End of file *****/

```

21.3.2 CRYPTO_Bench_PointMul.c

This application benchmarks elliptic curve point multiplication that is the foundation of ECDSA signatures and ECDH key agreement.

21.3.2.1 Example output

Point Multiplication Benchmark compiled Mar 19 2018 16:36:43

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
 System: Processor speed = 200.000 MHz
 Config: Static heap size = 36112 bytes
 Config: CRYPTO_VERSION = 22400 [2.24]
 Config: CRYPTO_MPI_BITS_PER_LIMB = 32

All times in ms

*** Prime curves ***

Algorithm	secp192r1	secp224r1	secp256r1	secp384r1	secp521r1
Binary, Basic	41.89 1.00x	49.05 1.00x	72.66 1.00x	122.67 1.00x	218.53 1.00x
2b, FW	37.72 1.11x	43.94 1.12x	64.39 1.13x	109.03 1.13x	194.30 1.12x
3b, FW	35.12 1.19x	41.28 1.19x	59.35 1.22x	102.18 1.20x	182.96 1.19x
4b, FW	34.61 1.21x	39.72 1.23x	57.45 1.26x	96.81 1.27x	172.44 1.27x
5b, FW	36.08 1.16x	41.26 1.19x	58.77 1.24x	97.10 1.26x	170.63 1.28x
6b, FW	41.71 1.00x	46.77 1.05x	65.56 1.11x	103.03 1.19x	178.11 1.23x
2b, RM	37.66 1.11x	43.93 1.12x	64.36 1.13x	108.53 1.13x	192.71 1.13x
3b, RM	34.93 1.20x	40.79 1.20x	59.04 1.23x	101.18 1.21x	180.81 1.21x
4b, RM	33.31 1.26x	38.59 1.27x	55.91 1.30x	94.81 1.29x	169.08 1.29x
5b, RM	33.54 1.25x	38.62 1.27x	54.98 1.32x	92.82 1.32x	164.57 1.33x
6b, RM	35.85 1.17x	40.74 1.20x	57.91 1.25x	94.42 1.30x	165.81 1.32x
2b, NAF	36.70 1.14x	43.13 1.14x	62.01 1.17x	106.09 1.16x	190.36 1.15x
3b, NAF	34.08 1.23x	39.59 1.24x	57.30 1.27x	98.21 1.25x	174.47 1.25x
4b, NAF	32.90 1.27x	38.28 1.28x	54.84 1.33x	92.84 1.32x	165.85 1.32x
5b, NAF	32.91 1.27x	37.86 1.30x	54.39 1.34x	91.60 1.34x	161.35 1.35x
6b, NAF	35.45 1.18x	40.14 1.22x	57.21 1.27x	93.80 1.31x	162.28 1.35x
Configured	41.87 1.00x	49.07 1.00x	72.68 1.00x	122.68 1.00x	218.64 1.00x

*** Koblitz curves ***

Algorithm	secp192k1	secp224k1	secp256k1
Binary, Basic	60.64 1.00x	81.30 1.00x	102.63 1.00x
2b, FW	54.58 1.11x	72.75 1.12x	90.91 1.13x
3b, FW	50.88 1.19x	68.30 1.19x	84.13 1.22x
4b, FW	50.14 1.21x	65.67 1.24x	81.76 1.26x
5b, FW	52.14 1.16x	68.22 1.19x	83.20 1.23x
6b, FW	60.07 1.01x	77.13 1.05x	92.41 1.11x
2b, RM	54.44 1.11x	72.55 1.12x	90.62 1.13x
3b, RM	50.44 1.20x	67.41 1.21x	83.55 1.23x
4b, RM	48.25 1.26x	63.95 1.27x	79.32 1.29x
5b, RM	48.56 1.25x	63.96 1.27x	77.96 1.32x
6b, RM	51.80 1.17x	67.36 1.21x	82.05 1.25x
2b, NAF	52.83 1.15x	71.06 1.14x	87.48 1.17x
3b, NAF	49.24 1.23x	65.35 1.24x	81.16 1.26x
4b, NAF	47.59 1.27x	63.18 1.29x	77.78 1.32x
5b, NAF	47.67 1.27x	62.61 1.30x	77.33 1.33x
6b, NAF	51.15 1.19x	66.30 1.23x	81.03 1.27x
Configured	60.67 1.00x	81.32 1.00x	102.68 1.00x

*** Brainpool curves ***

Algorithm	brainpoolP224r1	brainpoolP256r1	brainpoolP320r1	brainpoolP384r1	brainpoolP512r1
Binary, Basic	85.68 1.00x	112.23 1.00x	167.79 1.00x	261.45 1.00x	471.61 1.00x
2b, FW	76.94 1.11x	99.89 1.12x	150.63 1.11x	232.37 1.13x	418.08 1.13x
3b, FW	72.47 1.18x	92.76 1.21x	141.98 1.18x	219.27 1.19x	399.55 1.18x
4b, FW	69.76 1.23x	90.16 1.24x	135.34 1.24x	208.62 1.25x	379.56 1.24x
5b, FW	72.44 1.18x	91.77 1.22x	136.37 1.23x	209.01 1.25x	374.58 1.26x
6b, FW	81.32 1.05x	101.40 1.11x	147.86 1.13x	220.04 1.19x	386.95 1.22x
2b, RM	76.75 1.12x	99.64 1.13x	149.89 1.12x	230.81 1.13x	413.53 1.14x
3b, RM	71.58 1.20x	92.19 1.22x	140.44 1.19x	216.81 1.21x	394.34 1.20x
4b, RM	68.17 1.26x	87.81 1.28x	132.66 1.26x	204.54 1.28x	373.05 1.26x
5b, RM	68.16 1.26x	86.31 1.30x	130.49 1.29x	200.64 1.30x	364.81 1.29x
6b, RM	71.56 1.20x	90.63 1.24x	134.66 1.25x	203.95 1.28x	367.15 1.28x
2b, NAF	75.18 1.14x	96.10 1.17x	145.75 1.15x	224.42 1.17x	405.87 1.16x
3b, NAF	69.51 1.23x	89.62 1.25x	135.44 1.24x	210.20 1.24x	380.34 1.24x

4b, NAF	67.31 1.27x	86.06 1.30x	129.33 1.30x	200.35 1.30x	364.93 1.29x
5b, NAF	66.71 1.28x	85.65 1.31x	128.67 1.30x	198.18 1.32x	356.55 1.32x
6b, NAF	70.52 1.21x	89.66 1.25x	132.57 1.27x	202.19 1.29x	360.48 1.31x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
Configured	85.65 1.00x	112.03 1.00x	167.61 1.00x	261.20 1.00x	471.24 1.00x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
*** Twisted Brainpool curves ***					
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
Algorithm	brainpoolP224t1	brainpoolP256t1	brainpoolP320t1	brainpoolP384t1	brainpoolP512t1
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
Binary, Basic	79.49 1.00x	103.77 1.00x	154.57 1.00x	239.23 1.00x	429.11 1.00x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
2b, FW	70.87 1.12x	91.67 1.13x	137.65 1.12x	210.89 1.13x	376.61 1.14x
3b, FW	66.38 1.20x	84.55 1.23x	128.85 1.20x	197.86 1.21x	358.25 1.20x
4b, FW	63.62 1.25x	81.83 1.27x	122.33 1.26x	187.04 1.28x	337.65 1.27x
5b, FW	66.28 1.20x	83.49 1.24x	123.36 1.25x	187.42 1.28x	333.82 1.29x
6b, FW	75.25 1.06x	93.18 1.11x	134.87 1.15x	198.72 1.20x	345.64 1.24x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
2b, RM	70.67 1.12x	91.32 1.14x	137.04 1.13x	209.26 1.14x	372.13 1.15x
3b, RM	65.43 1.21x	83.78 1.24x	127.28 1.21x	195.23 1.23x	352.00 1.22x
4b, RM	61.99 1.28x	79.46 1.31x	119.62 1.29x	183.15 1.31x	332.01 1.29x
5b, RM	62.06 1.28x	78.10 1.33x	117.47 1.32x	179.20 1.34x	323.53 1.33x
6b, RM	65.38 1.22x	82.26 1.26x	121.51 1.27x	182.32 1.31x	325.77 1.32x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
2b, NAF	69.15 1.15x	87.98 1.18x	133.18 1.16x	202.81 1.18x	364.32 1.18x
3b, NAF	63.35 1.25x	81.27 1.28x	122.48 1.26x	188.65 1.27x	338.62 1.27x
4b, NAF	61.22 1.30x	77.82 1.33x	116.36 1.33x	178.53 1.34x	323.86 1.32x
5b, NAF	60.62 1.31x	77.31 1.34x	115.59 1.34x	176.41 1.36x	315.51 1.36x
6b, NAF	64.33 1.24x	81.29 1.28x	119.46 1.29x	180.25 1.33x	319.18 1.34x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				
Configured	79.60 1.00x	103.85 1.00x	154.72 1.00x	239.53 1.00x	429.60 1.00x
+-----+-----+-----+-----+-----+-----+	+-----+-----+-----+-----+-----+-----+				

Benchmark complete

21.3.2.2 Complete listing

```

/***** (c) SEGGER Microcontroller GmbH *****
*          The Embedded Experts
*          www.segger.com
***** ----- END-OF-HEADER ----- *****

File      : CRYPTO_Bench_PointMul.c
Purpose   : Benchmark EC point multiplication.

*/
/***** #include section *****
*/
#include "CRYPTO.h"
#include "SEGGER_MEM.h"
#include "SEGGER_SYS.h"

/***** Defines, configurable *****
*/
#define MAX_CHUNKS           244

/***** Defines, fixed *****
*/

```

```

#define CRYPTO_ASSERT(X) { if (!(X)) { CRYPTO_PANIC(); } }

#define CRYPTO_CHECK(X) /*lint -e{717,801,9036}
/* do { if ((Status = (X)) < 0) goto Finally; } while (0)

***** Local data types *****
*/
typedef CRYPTO_MPI_LIMB MPI_UNIT[CRYPTO_MPI_LIMBS_REQUIRED(2*521+63)+2];

typedef int (*POINTMUL_FUNC)
(CRYPTO_EC_POINT *pSelf, const CRYPTO_MPI *pK, const CRYPTO_EC_CURVE *pCurve, CRYPTO_MEM_CONTEXT *pMem);

typedef struct {
    const char * pText;           // Description of algorithm
    POINTMUL_FUNC pfPointMul;
} BENCH_ALG;

typedef struct {
    const CRYPTO_EC_CURVE * pCurve;
    const char * sName;
} BENCH_KEY;

***** Prototypes *****
*/
/*
*      Benchmark parameterization.
*/
static const BENCH_ALG _aBenchAlgs[] = {
    { "Binary", Basic }, CRYPTO_EC_Mul_Basic },
    { NULL, NULL },
    { "2b, FW", CRYPTO_EC_Mul_2b_FW },
    { "3b, FW", CRYPTO_EC_Mul_3b_FW },
    { "4b, FW", CRYPTO_EC_Mul_4b_FW },
    { "5b, FW", CRYPTO_EC_Mul_5b_FW },
    { "6b, FW", CRYPTO_EC_Mul_6b_FW },
    { NULL, NULL },
    { "2b, RM", CRYPTO_EC_Mul_2b_RM },
    { "3b, RM", CRYPTO_EC_Mul_3b_RM },
    { "4b, RM", CRYPTO_EC_Mul_4b_RM },
    { "5b, RM", CRYPTO_EC_Mul_5b_RM },
    { "6b, RM", CRYPTO_EC_Mul_6b_RM },
    { NULL, NULL },
    { "2b, NAF", CRYPTO_EC_Mul_2w_NAF },
    { "3b, NAF", CRYPTO_EC_Mul_3w_NAF },
    { "4b, NAF", CRYPTO_EC_Mul_4w_NAF },
    { "5b, NAF", CRYPTO_EC_Mul_5w_NAF },
    { "6b, NAF", CRYPTO_EC_Mul_6w_NAF },
    { NULL, NULL },
    { "Configured", CRYPTO_EC_Mul }
};

static const CRYPTO_EC_CURVE * _aPrimeCurves[] = {
    &CRYPTO_EC_CURVE_secp192r1,
    &CRYPTO_EC_CURVE_secp224r1,
    &CRYPTO_EC_CURVE_secp256r1,
    &CRYPTO_EC_CURVE_secp384r1,
    &CRYPTO_EC_CURVE_secp521r1
};

static const CRYPTO_EC_CURVE * _aKoblitzCurves[] = {
    &CRYPTO_EC_CURVE_secp192k1,
    &CRYPTO_EC_CURVE_secp224k1,
    &CRYPTO_EC_CURVE_secp256k1
};

```

```

static const CRYPTO_EC_CURVE * _aBrainpoolCurves[] = {
//&CRYPTO_EC_CURVE_brainpoolP160r1,
//&CRYPTO_EC_CURVE_brainpoolP192r1,
&CRYPTO_EC_CURVE_brainpoolP224r1,
&CRYPTO_EC_CURVE_brainpoolP256r1,
&CRYPTO_EC_CURVE_brainpoolP320r1,
&CRYPTO_EC_CURVE_brainpoolP384r1,
&CRYPTO_EC_CURVE_brainpoolP512r1
};

static const CRYPTO_EC_CURVE * _aBrainpoolTwistedCurves[] = {
//&CRYPTO_EC_CURVE_brainpoolP160t1,
//&CRYPTO_EC_CURVE_brainpoolP192t1,
&CRYPTO_EC_CURVE_brainpoolP224t1,
&CRYPTO_EC_CURVE_brainpoolP256t1,
&CRYPTO_EC_CURVE_brainpoolP320t1,
&CRYPTO_EC_CURVE_brainpoolP384t1,
&CRYPTO_EC_CURVE_brainpoolP512t1
};

//*********************************************************************
*
*      Static data
*
*****
```

```

static MPI_UNIT           _aUnits[MAX_CHUNKS];
static SEGGER_MEM_CONTEXT _MemContext;
static SEGGER_MEM_SELFTEST_HEAP _Heap;
static int                _ShowMemory = 0;
static unsigned           _AlgIndex;
static unsigned           _KeyIndex;
static float              _aBaseline[SEGGER_COUNTOF(_aBrainpoolCurves)];
```

```

//*********************************************************************
*
*      Static code
*
*****
```

```

*_ConvertTicksToSeconds()

* Function description
*   Convert ticks to seconds.
*
* Parameters
*   Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
*
* Return value
*   Number of seconds corresponding to tick.
*/
static float _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0f;
}

//*********************************************************************
*
*      _BenchmarkSinglePointMul()

* Function description
*   Count the number of point multiplications completed in one second.
*
* Parameters
*   pfPointMul - Pointer to point multiply implementation.
*   pCurve     - Pointer to EC group.
*/
static void _BenchmarkSinglePointMul(POINTMUL_FUNC pfPointMul, const CRYPTO_EC_CURVE *pCurve) {
    CRYPTO_EC_POINT Point;
    CRYPTO_MPI     Scalar;
    U64           OneSecond;
    U64           T0;
```

```

U64          Elapsed;
int           Loops;
int           Status;
unsigned      PeakBytes;
float         Multiplier;
float         Time;
//
PeakBytes = 0;
Loops     = 0;
//
CRYPTO_EC_InitPoint(&Point, &_MemContext);
CRYPTO_MPI_Init    (&Scalar, &_MemContext);
//
CRYPTO_EC_AssignPoint(&Point, &pCurve->G);
CRYPTO_EC_MakeProjective(&Point);
CRYPTO_MPI_LoadHex(&Scalar, "3243F6A8885A308D313198A2E03707344A4093822299F31D0082EFA98EC4E6C89452821E638D0
// Hex digits of Pi
CRYPTO_MPI_TrimBits(&Scalar, CRYPTO_MPI_BitCount(&pCurve->P)-1);
//
// Count number of modular exponentiations completed in 1s.
//
OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
T0 = SEGGER_SYS_OS_GetTimer();
do {
    _Heap.Stats.NumInUseMax = _Heap.Stats.NumInUse;
    CRYPTO_CHECK(pfPointMul(&Point, &Scalar, pCurve, &_MemContext));
    PeakBytes = _Heap.Stats.NumInUseMax * sizeof(MPI_UNIT);
    ++Loops;
    Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
} while (Elapsed < OneSecond);
//
Finally:
Elapsed = SEGGER_SYS_OS_GetTimer() - T0;
//
CRYPTO_EC_KillPoint(&Point);
CRYPTO_MPI_Kill(&Scalar);
//
Time = 1000.0f * _ConvertTicksToSeconds(Elapsed) / Loops;
if (_AlgIndex == 0) {
    _aBaseline[_KeyIndex] = Time;
}
Multiplier = _aBaseline[_KeyIndex] / (float)Time;
if (Status < 0) {
    SEGGER_SYS_IO_Printf("      -Fail- | ");
} else {
    if (_ShowMemory) {
        SEGGER_SYS_IO_Printf("      %8d | ", PeakBytes);
    } else {
        SEGGER_SYS_IO_Printf("%9.2f %4.2fx | ", Time, Multiplier);
    }
}
}

/****************************************
*
*      _PrintFooter()
*
*  Function description
*      Print column footer for table.
*
*  Parameters
*      NumCurves - Number of curves in group.
*/
static void _PrintFooter(unsigned NumCurves) {
    unsigned i;
    //
    SEGGER_SYS_IO_Printf("+-----+");
    for (i = 0; i < NumCurves; ++i) {
        SEGGER_SYS_IO_Printf("-----+");
    }
    SEGGER_SYS_IO_Printf("\n");
}

/****************************************
*
*      _PrintHeader()

```

```

/*
 * Function description
 *   Print column headers for table.
 *
 * Parameters
 *   ppCurve - Pointer to curve group array.
 *   NumCurves - Number of curves in group.
 */
static void _PrintHeader(const CRYPTO_EC_CURVE * const * const ppCurve, unsigned NumCurves) {
    unsigned i;
    //
    SEGGER_SYS_IO_Printf("+-----+");
    for (i = 0; i < NumCurves; ++i) {
        SEGGER_SYS_IO_Printf("-----+");
    }
    SEGGER_SYS_IO_Printf("\n");
    SEGGER_SYS_IO_Printf(" | Algorithm | ");
    for (i = 0; i < NumCurves; ++i) {
        SEGGER_SYS_IO_Printf("%15s | ", ppCurve[i]->aCurveName);
    }
    SEGGER_SYS_IO_Printf("\n");
    SEGGER_SYS_IO_Printf("+-----+");
    for (i = 0; i < NumCurves; ++i) {
        SEGGER_SYS_IO_Printf("-----+");
    }
    SEGGER_SYS_IO_Printf("\n");
}

/***********************/

/*
 *      _BenchmarkGroup()
 *
 * Function description
 *   Benchmark a group of curves.
 *
 * Parameters
 *   sGroupName - Zero-terminate group name.
 *   pCurve     - Pointer to curve group array.
 *   NumCurves - Number of curves in group.
 */
static void _BenchmarkGroup(const char *sGroupName, const CRYPTO_EC_CURVE * const * const pCurve, unsigned N
    SEGGER_SYS_IO_Printf("*** %s ***\n\n", sGroupName);
    //
    _PrintHeader(pCurve, NumCurves);
    for (_AlgIndex = 0; _AlgIndex < SEGGER_COUNTOF(_aBenchAlgs); ++_AlgIndex) {
        if (_aBenchAlgs[_AlgIndex].pfPointMul == NULL) {
            _PrintFooter(NumCurves);
        } else {
            SEGGER_SYS_IO_Printf(" | %-15s | ", _aBenchAlgs[_AlgIndex].pText);
            for (_KeyIndex = 0; _KeyIndex < NumCurves; ++_KeyIndex) {
                _BenchmarkSinglePointMul(_aBenchAlgs[_AlgIndex].pfPointMul, pCurve[_KeyIndex]);
            }
            SEGGER_SYS_IO_Printf("\n");
        }
    }
    _PrintFooter(NumCurves);
    SEGGER_SYS_IO_Printf("\n");
}

/***********************/

/*
 *      Public code
 *
 ****
 */

/*
 *      MainTask()
 *
 * Function description
 *   Main entry point for application to run all the tests.
 */
void MainTask(void);
void MainTask(void) {
    //
}

```

```
CRYPTO_Init();
SEGGER_SYS_Init();
SEGGER_MEM_SELFTEST_HEAP_Init(&_MemContext, &_Heap, _aUnits, MAX_CHUNKS, sizeof(MPI_UNIT));
//SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
SEGGER_SYS_IO_Printf("Point Multiplication Benchmark compiled " __DATE__ " " __TIME__ "\n
\r\n");
//SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
if (SEGGER_SYS_GetProcessorSpeed() > 0) {
    SEGGER_SYS_IO_Printf("System: Processor speed      = %.3f MHz
\r\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
}
SEGGER_SYS_IO_Printf("Config: Static heap size      = %u bytes\n", sizeof(_aUnits));
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION      = %u
[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
SEGGER_SYS_IO_Printf("Config: CRYPTO_MPI_BITS_PER_LIMB = %u
\r\n", CRYPTO_MPI_BITS_PER_LIMB);
SEGGER_SYS_IO_Printf("\n");
//
SEGGER_SYS_IO_Printf("All times in ms\n");
SEGGER_SYS_IO_Printf("\n");
//
_BenchmarkGroup("Prime curves", _aPrimeCurves,
SEGGER_COUNTOF(_aPrimeCurves));
_BenchmarkGroup("Koblitz curves", _aKoblitzCurves,
SEGGER_COUNTOF(_aKoblitzCurves));
_BenchmarkGroup("Brainpool curves", _aBrainpoolCurves,
SEGGER_COUNTOF(_aBrainpoolCurves));
_BenchmarkGroup("Twisted Brainpool
curves", _aBrainpoolTwistedCurves, SEGGER_COUNTOF(_aBrainpoolTwistedCurves));
//
SEGGER_SYS_IO_Printf("Benchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}

***** End of file *****
```

21.4 Random bits

21.4.1 CRYPTO_Bench_RNG.c

This application benchmarks the configured performance of Fortuna, Hash-DRBG and HMAC_DRBG random bit generators.

21.4.1.1 Example output

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
RNG Benchmark compiled Mar 19 2018 16:42:20

Compiler: clang 5.0.0 (tags/RELEASE_500/final)
System: Processor speed          = 200.000 MHz
Config: CRYPTO_VERSION          = 22400 [2.24]
Config: CRYPTO_CONFIG_SHA1_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA256_OPTIMIZE = 1
Config: CRYPTO_CONFIG_SHA512_OPTIMIZE = 2
Config: CRYPTO_CONFIG_DES_OPTIMIZE = 5
Config: CRYPTO_CONFIG_AES_OPTIMIZE = 7

RNG Performance
=====

+-----+-----+
| Algorithm           | MB/s |
+-----+-----+
| Fortuna             | 11.09 |
+-----+-----+
| Hash_DRBG-SHA-1    | 1.72  |
| Hash_DRBG-SHA-224   | 2.30  |
| Hash_DRBG-SHA-256   | 2.77  |
| Hash_DRBG-SHA-384   | 0.49  |
| Hash_DRBG-SHA-512   | 0.67  |
| Hash_DRBG-SHA-512/224 | 0.29  |
| Hash_DRBG-SHA-512/256 | 0.33  |
+-----+-----+
| HMAC_DRBG-SHA-1     | 0.66  |
| HMAC_DRBG-SHA-224   | 0.90  |
| HMAC_DRBG-SHA-256   | 1.02  |
| HMAC_DRBG-SHA-384   | 0.13  |
| HMAC_DRBG-SHA-512   | 0.17  |
| HMAC_DRBG-SHA-512/224 | 0.08  |
| HMAC_DRBG-SHA-512/256 | 0.09  |
+-----+-----+
| CTR-DRBG-TDES       | 4.38  |
| CTR-DRBG-AES-128    | 10.04 |
| CTR-DRBG-AES-192    | 10.10 |
| CTR-DRBG-AES-256    | 9.85  |
+-----+-----+

Benchmark complete
```

21.4.1.2 Complete listing

```
*****
* (c) SEGGER Microcontroller GmbH
*      The Embedded Experts
*      www.segger.com
*****
----- END-OF-HEADER -----
File      : CRYPTO_Bench_RNG.c
Purpose   : Benchmark RNG (DRBG) implementation.
```

```
/*
 ****
 *      #include section
 *
 ****
 */

#include "CRYPTO.h"
#include "SEGGER_SYS.h"

/*
 *
 *      Static data
 *
 ****
 */

static U8           _aTestMessage[65536];
static CRYPTO_FORTUNA_CONTEXT _Fortuna;

/*
 *
 *      Static code
 *
 ****
 */

/*
 *
 *      _FortunaInit()
 *
 *      Function description
 *      Initialize Fortuna.
 *
 *      Additional information
 *      We are benchmarking only performance, not quality of random
 *      data, so just seed with fixed data so that benchmark is
 *      repeatable.
 */
static void _FortunaInit(void) {
    static const U8 aFixedData[] = { 'C', 'R', 'Y', 'P', 'T', 'O', '!' };
    //
    CRYPTO_FORTUNA_Init(&_Fortuna);
    while (CRYPTO_FORTUNA_Status(&_Fortuna) < 0) {
        CRYPTO_FORTUNA_Add(&_Fortuna, 0, &aFixedData[0], sizeof(aFixedData));
    }
}

/*
 *
 *      _FortunaGet()
 *
 *      Function description
 *      Get data from Fortuna.
 *
 *      Parameters
 *      pOutput - Pointer to octet string that receives the random data.
 *      OutputLen - Octet length of the octet string.
 */
static void _FortunaGet(U8 *pOutput, unsigned OutputLen) {
    CRYPTO_FORTUNA_Get(&_Fortuna, pOutput, OutputLen);
}

/*
 *
 *      _ConvertTicksToSeconds()
 *
 *      Function description
 *      Convert ticks to seconds.
 *
 *      Parameters
 *      Ticks - Number of ticks reported by SEGGER_SYS_OS_GetTimer().
 *
```

```

*   Return value
*   Number of seconds corresponding to tick.
*/
static double _ConvertTicksToSeconds(U64 Ticks) {
    return SEGGER_SYS_OS_ConvertTicksToMicros(Ticks) / 1000000.0;
}

*****
*
*     _RNG_Benchmark()
*
* Function description
*   Benchmarks a RNG implementation.
*
* Parameters
*   sAlgorithm - RNG algorithm name.
*   pAPI       - Pointer to RNG API.
*/
static void _RNG_Benchmark(const char *sAlgorithm, const CRYPTO_RNG_API *pAPI) {
    U64          T0;
    U64          OneSecond;
    unsigned      n;
    //
    SEGGER_SYS_IO_Printf(" | %-21s | ", sAlgorithm);
    OneSecond = SEGGER_SYS_OS_ConvertMicrosToTicks(1000000);
    //
    // ECB encrypt
    //
    T0 = SEGGER_SYS_OS_GetTimer();
    n = 0;
    if (pAPI->pfInit != NULL) {
        pAPI->pfInit();
    }
    while (SEGGER_SYS_OS_GetTimer() - T0 < OneSecond) {
        pAPI->pfGet(&_aTestMessage[0], sizeof(_aTestMessage));
        n += sizeof(_aTestMessage);
    }
    T0 = SEGGER_SYS_OS_GetTimer() - T0;
    SEGGER_SYS_IO_Printf("%7.2f |\n", (double)n / (1024.0*1024.0) / _ConvertTicksToSeconds(T0));
}

*****
*
*     Public code
*
*****
*/

```

```

*****
*
*     MainTask()
*
* Function description
*   Main entry point for application to run all the tests.
*/
void MainTask(void);
void MainTask(void) {
    //
    const CRYPTO_RNG_API *pEntropyAPI;
    const CRYPTO_RNG_API *pRngAPI;
    //
    static const CRYPTO_RNG_API _FortunaMethods = {
        _FortunaInit,
        _FortunaGet,
        NULL,
        NULL
    };
    //
    CRYPTO_Init();
    SEGGER_SYS_Init();
    //
    CRYPTO_RNG_QueryInstallEx(&pRngAPI, &pEntropyAPI);
    if (pEntropyAPI == NULL) {
        SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
        SEGGER_SYS_OS_PauseBeforeHalt();
    }
}

```

```

    SEGGER_SYS_OS_Halt(100);
}
//SEGGER_SYS_IO_Printf("%s      www.segger.com\n", CRYPTO_GetCopyrightText());
SEGGER_SYS_IO_Printf("RNG Benchmark compiled " __DATE__ " " __TIME__ "\n\n");
//SEGGER_SYS_IO_Printf("Compiler: %s\n", SEGGER_SYS_GetCompiler());
if (SEGGER_SYS_GetProcessorSpeed() > 0) {
    SEGGER_SYS_IO_Printf("System: Processor speed = %.3f MHz
\n", (double)SEGGER_SYS_GetProcessorSpeed() / 1000000.0f);
}
SEGGER_SYS_IO_Printf("Config: CRYPTO_VERSION = %u
[%s]\n", CRYPTO_VERSION, CRYPTO_GetVersionText());
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA1_OPTIMIZE = %d
\n", CRYPTO_CONFIG_SHA1_OPTIMIZE);
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA256_OPTIMIZE = %d
\n", CRYPTO_CONFIG_SHA256_OPTIMIZE);
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_SHA512_OPTIMIZE = %d
\n", CRYPTO_CONFIG_SHA512_OPTIMIZE);
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_DES_OPTIMIZE = %d
\n", CRYPTO_CONFIG_DES_OPTIMIZE);
SEGGER_SYS_IO_Printf("Config: CRYPTO_CONFIG_AES_OPTIMIZE = %d
\n", CRYPTO_CONFIG_AES_OPTIMIZE);
SEGGER_SYS_IO_Printf("\n");
//SEGGER_SYS_IO_Printf("RNG Performance\n");
SEGGER_SYS_IO_Printf("=====+\n\n");
SEGGER_SYS_IO_Printf("-----+-----+\n");
SEGGER_SYS_IO_Printf(" | Algorithm | MB/s |\n");
SEGGER_SYS_IO_Printf("-----+-----+\n");
//_RNG_Benchmark("Entropy source", pEntropyAPI);
SEGGER_SYS_IO_Printf("-----+-----+\n");
_RNG_Benchmark("Fortuna", &FortunaMethods);
SEGGER_SYS_IO_Printf("-----+-----+\n");
if (CRYPTO_SHA1_IsInstalled()) {
    _RNG_Benchmark("Hash_DRBG-SHA-1", &CRYPTO_RNG_DRBG_HASH_SHA1);
}
if (CRYPTO_SHA224_IsInstalled()) {
    _RNG_Benchmark("Hash_DRBG-SHA-224", &CRYPTO_RNG_DRBG_HASH_SHA224);
}
if (CRYPTO_SHA256_IsInstalled()) {
    _RNG_Benchmark("Hash_DRBG-SHA-256", &CRYPTO_RNG_DRBG_HASH_SHA256);
}
if (CRYPTO_SHA512_IsInstalled()) {
    _RNG_Benchmark("Hash_DRBG-SHA-384", &CRYPTO_RNG_DRBG_HASH_SHA384);
    _RNG_Benchmark("Hash_DRBG-SHA-512", &CRYPTO_RNG_DRBG_HASH_SHA512);
    _RNG_Benchmark("Hash_DRBG-SHA-512/224", &CRYPTO_RNG_DRBG_HASH_SHA512_224);
    _RNG_Benchmark("Hash_DRBG-SHA-512/256", &CRYPTO_RNG_DRBG_HASH_SHA512_256);
}
SEGGER_SYS_IO_Printf("-----+-----+\n");
if (CRYPTO_SHA1_IsInstalled()) {
    _RNG_Benchmark("HMAC_DRBG-SHA-1", &CRYPTO_RNG_DRBG_HMAC_SHA1);
}
if (CRYPTO_SHA224_IsInstalled()) {
    _RNG_Benchmark("HMAC_DRBG-SHA-224", &CRYPTO_RNG_DRBG_HMAC_SHA224);
}
if (CRYPTO_SHA256_IsInstalled()) {
    _RNG_Benchmark("HMAC_DRBG-SHA-256", &CRYPTO_RNG_DRBG_HMAC_SHA256);
}
if (CRYPTO_SHA512_IsInstalled()) {
    _RNG_Benchmark("HMAC_DRBG-SHA-384", &CRYPTO_RNG_DRBG_HMAC_SHA384);
    _RNG_Benchmark("HMAC_DRBG-SHA-512", &CRYPTO_RNG_DRBG_HMAC_SHA512);
    _RNG_Benchmark("HMAC_DRBG-SHA-512/224", &CRYPTO_RNG_DRBG_HMAC_SHA512_224);
    _RNG_Benchmark("HMAC_DRBG-SHA-512/256", &CRYPTO_RNG_DRBG_HMAC_SHA512_256);
}
SEGGER_SYS_IO_Printf("-----+-----+\n");
if (CRYPTO_TDES_IsInstalled()) {
    _RNG_Benchmark("CTR-DRBG-TDES", &CRYPTO_RNG_DRBG_CTR_TDES);
}
if (CRYPTO_AES_IsInstalled()) {
    _RNG_Benchmark("CTR-DRBG-AES-128", &CRYPTO_RNG_DRBG_CTR_AES128);
    _RNG_Benchmark("CTR-DRBG-AES-192", &CRYPTO_RNG_DRBG_CTR_AES192);
    _RNG_Benchmark("CTR-DRBG-AES-256", &CRYPTO_RNG_DRBG_CTR_AES256);
}
//
```

```
SEGGER_SYS_IO_Printf( "+-----+\n" );
//  
SEGGER_SYS_IO_Printf("\nBenchmark complete\n");
SEGGER_SYS_OS_PauseBeforeHalt();
SEGGER_SYS_OS_Halt(0);
}  
  
***** End of file *****
```

Chapter 22

Resource use

22.1 Context sizes

The application `CRYPTO_DumpContextSize` prints the context sizes for hash algorithms, ciphers, and random bit generators.

22.1.1 Example output - minimum size

The following output is for a minimum size configuration on a Cortex-M device:

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
Dump Context Size 2.24 compiled Mar 19 2018 16:43:18
```

```
Compiler: clang 5.0.0 (tags/RELEASE_500/final)
```

Algorithm	Size	Configuration	=	0
DES	392	CRYPTO_CONFIG_DES_OPTIMIZE	=	0
AES	248	CRYPTO_CONFIG_AES_OPTIMIZE	=	0
ARIA	280	CRYPTO_CONFIG_ARIA_OPTIMIZE	=	0
Camellia	280	CRYPTO_CONFIG_CAMELLIA_OPTIMIZE	=	0
CAST	136	CRYPTO_CONFIG_CAST_OPTIMIZE	=	0
Blowfish	4172	CRYPTO_CONFIG_BLOWFISH_OPTIMIZE	=	0
Twofish	200	CRYPTO_CONFIG_TWOFISH_OPTIMIZE	=	0
RC4	258			
MD5	104	CRYPTO_CONFIG_MD5_OPTIMIZE	=	0
RIPEMD160	112	CRYPTO_CONFIG_RIPEMD160_OPTIMIZE	=	0
SHA-1	112	CRYPTO_CONFIG_SHA1_OPTIMIZE	=	0
SHA-256	120	CRYPTO_CONFIG_SHA256_OPTIMIZE	=	0
SHA-512	216	CRYPTO_CONFIG_SHA512_OPTIMIZE	=	0
SHA3-224	232	CRYPTO_CONFIG_SHA3_OPTIMIZE	=	0
SHA3-256	232			
SHA3-384	232			
SHA3-512	232			
HMAC-MD5	232			
HMAC-RIPEMD160	240			
HMAC-SHA-1	240			
HMAC-SHA-256	248			
HMAC-SHA-512	472			
HMAC-SHA-512/224	472			
HMAC-SHA-512/256	472			
HMAC-SHA3-224	520			
HMAC-SHA3-256	504			
HMAC-SHA3-384	440			
HMAC-SHA3-512	376			
CMAC-AES	320			
CMAC-TDES	432			
CMAC-CAST	176			
CMAC-SEED	204			
CMAC-ARIA	352			
CMAC-Camellia	352			
CMAC-Twofish	352			
CMAC-Blowfish	352			
GMAC-AES	328			
GMAC-SEED	216			
GMAC-ARIA	360			
GMAC-Camellia	360			
GMAC-Twofish	280			
Poly1305-AES	80			
Poly1305-SEED	80			

Poly1305-ARIA	80	
Poly1305-Camellia	80	
Poly1305-Twofish	80	
<hr/>		
DRBG_Hash-SHA-1	116	
DRBG_Hash-SHA-256	116	
DRBG_Hash-SHA-512	228	
DRBG_HMAC-SHA-1	44	
DRBG_HMAC-SHA-256	68	
DRBG_HMAC-SHA-512	132	
<hr/>		
Dump complete.		

22.1.2 Example output - maximum speed

The following output is for a maximum speed configuration on a Cortex-M device:

```
Copyright (c) 2014-2018 SEGGER Microcontroller GmbH      www.segger.com
Dump Context Size 2.24 compiled Mar 19 2018 16:43:13
```

```
Compiler: clang 5.0.0 (tags/RELEASE_500/final)
```

Algorithm	Size	Configuration	=	5
DES	392	CRYPTO_CONFIG_DES_OPTIMIZE	=	5
AES	248	CRYPTO_CONFIG_AES_OPTIMIZE	=	7
ARIA	280	CRYPTO_CONFIG_ARIA_OPTIMIZE	=	1
Camellia	280	CRYPTO_CONFIG_CAMELLIA_OPTIMIZE	=	3
CAST	136	CRYPTO_CONFIG_CAST_OPTIMIZE	=	1
Blowfish	4172	CRYPTO_CONFIG_BLOWFISH_OPTIMIZE	=	1
Twofish	4260	CRYPTO_CONFIG_TWOFISH_OPTIMIZE	=	15
RC4	258			
<hr/>				
MD5	104	CRYPTO_CONFIG_MD5_OPTIMIZE	=	1
RIPEMD160	112	CRYPTO_CONFIG_RIPEMD160_OPTIMIZE	=	1
SHA-1	112	CRYPTO_CONFIG_SHA1_OPTIMIZE	=	1
SHA-256	120	CRYPTO_CONFIG_SHA256_OPTIMIZE	=	1
SHA-512	216	CRYPTO_CONFIG_SHA512_OPTIMIZE	=	2
SHA3-224	232	CRYPTO_CONFIG_SHA3_OPTIMIZE	=	1
SHA3-256	232			
SHA3-384	232			
SHA3-512	232			
<hr/>				
HMAC-MD5	232			
HMAC-RIPEMD160	240			
HMAC-SHA-1	240			
HMAC-SHA-256	248			
HMAC-SHA-512	472			
HMAC-SHA-512/224	472			
HMAC-SHA-512/256	472			
HMAC-SHA3-224	520			
HMAC-SHA3-256	504			
HMAC-SHA3-384	440			
HMAC-SHA3-512	376			
<hr/>				
CMAC-AES	320			
CMAC-TDES	432			
CMAC-CAST	176			
CMAC-SEED	204			
CMAC-ARIA	352			
CMAC-Camellia	352			
CMAC-Twofish	352			
CMAC-Blowfish	352			

GMAC-AES	328	
GMAC-SEED	216	
GMAC-ARIA	360	
GMAC-Camellia	360	
GMAC-Twofish	4344	
+	-----+-----+	
Poly1305-AES	80	
Poly1305-SEED	80	
Poly1305-ARIA	80	
Poly1305-Camellia	80	
Poly1305-Twofish	80	
+	-----+-----+	
DRBG_Hash-SHA-1	116	
DRBG_Hash-SHA-256	116	
DRBG_Hash-SHA-512	228	
DRBG_HMAC-SHA-1	44	
DRBG_HMAC-SHA-256	68	
DRBG_HMAC-SHA-512	132	
+	-----+-----+	

Dump complete.

Chapter 23

Bibliography

This section summarizes the national, international, and other standards that are relevant to emCrypt.

23.1 IETF RFCs

- IETF RFC 1321 — *The MD5 Message-Digest Algorithm*
- IETF RFC 2104 — *HMAC: Keyed-Hashing for Message Authentication*
- IETF RFC 3394 — *Advanced Encryption Standard (AES) Key Wrap Algorithm*
- IETF RFC 3713 — *A Description of the Camellia Encryption Algorithm*
- IETF RFC 3566 — *The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec*
- IETF RFC 4269 — *The SEED Encryption Algorithm*
- IETF RFC 5529 — *Modes of Operation for Camellia for Use with IPsec*
- IETF RFC 5649 — *Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*
- IETF RFC 5652 — *Cryptographic Message Syntax (CMS)*
- IETF RFC 5794 — *A Description of the ARIA Encryption Algorithm*
- IETF RFC 7539 — *ChaCha20 and Poly1305 for IETF Protocols*
- IETF RFC 7693 — *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*
- IETF RFC 2437 — *PKCS #1: RSA Cryptography Specifications Version 2.2*

23.2 ANSI standards

- ANS X9.63 — *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*

23.3 ISO standards

- ISO/IEC 18033-2 — *Encryption algorithms — Part 2: Asymmetric ciphers*

23.4 IEEE standards

- IEEE Std 1363-2000 — *IEEE Standard Specifications for Public-Key Cryptography*

23.5 NIST standards and special publications

- FIPS PUB 46-3 — *Data Encryption Standard (DES)*
- FIPS PUB 180-4 — *Secure Hash Standard (SHS)*
- FIPS PUB 186-4 — *Digital Signature Standard (DSS)*
- FIPS PUB 198-1 — *The Keyed-Hash Message Authentication Code (HMAC)*
- FIPS PUB 197 — *Specification for the Advanced Encryption Standard (AES)*
- FIPS PUB 202 — *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*
- NIST SP 800-38B — *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*
- NIST SP 800-38D — *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*
- NIST SP 800-38E — *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*
- NIST SP 800-38F — *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*
- NIST SP 800-90A — *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
- NIST SP 800-185 — *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*

23.6 Other relevant documents

- Antoon Bosselaers — *The hash function RIPEMD-160*
- Schneier et al — *Twofish: A 128-Bit Block Cipher*
- draft-sca-cfrg-sm3-02 — *The SM3 Cryptographic Hash Function*

Chapter 24

Glossary

3DES

Triple DES. A classical means to extend the 56-bit key space of DES to 112 bits by combining two 56-bit keys in three DES operations (two-key 3DES-EDE). 3DES is also known as TDES in standards documentation.

AEAD

Authenticated Encryption with Additional Data. A modern cipher mode that combines encryption with authentication where both can run in parallel and enhance throughput in hardware implementations. AES-GCM and AES-CCM are AEAD ciphers..

AES

Advanced Encryption Standard. A modern 128-bit block cipher, specified by NIST, that replaces the DES standard.

ASN.1

Abstract Syntax Notation 1. A specification of how to encode primitive data as octet streams.

CBC

Cipher Block Chaining. A cipher mode that uses the output of the previous block as an input to the following block to be encrypted.

CMS

Cryptographic Message Syntax. An IETF standard that specifies the format of cryptographic data in messages; defined by [RFC 5652](#).

DES

Data Encryption Standard. A retired 64-bit block cipher with 56-bit keys defined by NIST.

DH

Diffie-Hellman. A key agreement scheme based on discrete logarithm cryptography.

DHE

Ephemeral Diffie-Hellman. A key agreement scheme based on discrete logarithm cryptography where keys are generated once per connection and are unique for each connection. This guarantees Perfect Forward Secrecy (PFS).

DRBG

Deterministic Random Bit Generator. A random bit generator that will generate the same sequence of bits given the same seed as input, but the output is random using standard randomness tests.

DSA

Digital Signature Algorithm. The algorithm that signs a piece of data, specified in the Digital Signature Standard.

DSS

Digital Signature Standard. The NIST digital signature standard that specifies the Digital Signature Algorithm (DSA).

DTLS

Datagram Transport Layer Security. A scheme similar to TLS that transports TLS messages over UDP datagrams.

ECB

Electronic Code Book. An insecure mode for a block cipher where each block is encrypted in isolation and not chained.

ECC

Elliptic Curve Cryptography. Cryptography based on elliptic curves.

ECDH

Elliptic Curve Diffie-Hellman. The equivalent of Diffie-Hellman using elliptic curves.

ECDHE

Elliptic Curve Diffie-Hellman Ephemeral. As ECDH but using ephemeral keys. Provides perfect forward secrecy (PFS).

ECDSA

Elliptic Curve Digital Signature Algorithm. A standard for digital signatures signed using elliptic curve cryptography rather than discrete log cryptography. The elliptic curve analog of the discrete log signature scheme (DSA).

FIPS

Federal Information Processing Standard. A standard issued by NIST for Federal use and widely adopted throughout the world.

GCM

Galois Counter Mode. A modern mode for a block cipher where the authentication tag is computed using arithmetic in a Galois field, GF(2^{128})..

HMAC

Hashed Message Authentication Code. A MAC that is computed using a cryptographic hash function in combination with a secret key.

IANA

Internet Assigned Numbers Authority. IANA is responsible for the global coordination of the Internet protocol resources for TLS as part of its mandate.

ICV

Integrity Check Value. See MAC.

IETF

Internet Engineering Task Force. The IETF produces high quality, relevant technical documents that influence the way people design, use, and manage the Internet.

KAT

Known Answer Test. A test vector that has a deterministic, invariant known answer that an algorithm can be validated against.

MAC

Message Authentication Code. A small piece of information used to authenticate a message and to provide integrity and authenticity assurances about the content of the message.

MD5

Message Digest Algorithm 5. A MAC defined by RSA Data Security, Inc.

MPI

Multiprecision integer. An integer that can grow and shrink as required to represent cryptographic numbers.

NIST

National Institute of Standards and Technology. An organization in the USA responsible for the standardization of a wide range of technologies that pervade the IT industry.

PFS

Perfect Forward Secrecy. A means to ensure that exposure of the session keys for one connection and its decryption does not expose other sessions to subsequent decryption using the recovered cryptographic material.

PKI

Public Key Infrastructure. A set of specifications and mechanisms that can provide confidence in and interoperability of public key cryptography systems.

PRF

Pseudorandom Function. A function defined in the TLS specifications used to generate various unpredictable internal data used by TLS connections.

PSK

Preshared Key. A shared private key held by two entities agreed in advance of communication.

RFC

Request For Comment. The standard means that IETF disseminates Internet standards.

RNG

Random Number Generator. A device that generates true random numbers.

RSA

Rivest, Shamir, Adleman. The name of the cryptosystem based on Integer Factorization problems defined by the three authors.

SHA

Secure Hash Algorithm. The standard set of one-way functions that provide a message digest, as specified by NIST.

SSL

Secure Sockets Layer. Previous name for Transport Layer Security (TLS).

TDES

Triple DES. See 3DES.

TLS

Transport Layer Security. The current name and standard definition that provides confidential and authenticated transmission of data over insecure channels.

Chapter 25

Indexes

25.1 Index of functions

CRYPTO_AES_128_CBC_CAVS_SelfTest, **1240**
 CRYPTO_AES_128_CCM_CAVS_SelfTest, **1246**
 CRYPTO_AES_128_ECB_CAVS_SelfTest, **1243**
 CRYPTO_AES_128_GCM_CAVS_SelfTest, **1250**
 CRYPTO_AES_192_CBC_CAVS_SelfTest, **1241**
 CRYPTO_AES_192_CCM_CAVS_SelfTest, **1247**
 CRYPTO_AES_192_ECB_CAVS_SelfTest, **1244**
 CRYPTO_AES_192_GCM_CAVS_SelfTest, **1251**
 CRYPTO_AES_256_CBC_CAVS_SelfTest, **1242**
 CRYPTO_AES_256_CCM_CAVS_SelfTest, **1248**
 CRYPTO_AES_256_ECB_CAVS_SelfTest, **1245**
 CRYPTO_AES_256_GCM_CAVS_SelfTest, **1252**
 CRYPTO_AES_CBC_Decrypt, **1205**
 CRYPTO_AES_CBC_Encrypt, **1204**
 CRYPTO_AES_CCM_Decrypt, **1211**, 2757
 CRYPTO_AES_CCM_Encrypt, **1210**, 2757
 CRYPTO_AES_CCM_SP800x38C_SelfTest, **1249**
 CRYPTO_AES_CTR_Decrypt, **1209**
 CRYPTO_AES_CTR_Encrypt, **1208**
 CRYPTO_AES_Decrypt, **1201**
 CRYPTO_AES_ECB_Decrypt, **1203**
 CRYPTO_AES_ECB_Encrypt, 1139, 1139, **1202**
 CRYPTO_AES_Encrypt, **1200**
 CRYPTO_AES_GCM_Decrypt, **1213**, 2757
 CRYPTO_AES_GCM_Encrypt, **1212**, 2756
 CRYPTO_AES_InitDecrypt, **1198**
 CRYPTO_AES_InitEncrypt, 1139, **1197**
 CRYPTO_AES_Install, **1194**, 2649, 2651, 2652, 2654, 2654, 2655, 2656, 2657, 2659, 2659, 2660, 2662, 2665
 CRYPTO_AES_IsInstalled, **1195**, 2805
 CRYPTO_AES_Kill, 1139, **1199**
 CRYPTO_AES_OFB_Decrypt, **1207**
 CRYPTO_AES_OFB_Encrypt, **1206**
 CRYPTO_AES_QueryInstall, **1196**, 2757
 CRYPTO_AES_RFC3602_SelfTest, **1253**
 CRYPTO_AESKW_RFC3394_SelfTest, **2339**
 CRYPTO_ARIA_CBC_Decrypt, **1359**
 CRYPTO_ARIA_CBC_Encrypt, **1358**
 CRYPTO_ARIA_CCM_Decrypt, **1365**
 CRYPTO_ARIA_CCM_Encrypt, **1364**
 CRYPTO_ARIA_CTR_Decrypt, **1361**
 CRYPTO_ARIA_CTR_Encrypt, **1360**
 CRYPTO_ARIA_Decrypt, **1355**
 CRYPTO_ARIA_ECB_Decrypt, **1357**
 CRYPTO_ARIA_ECB_Encrypt, **1356**
 CRYPTO_ARIA_Encrypt, **1354**
 CRYPTO_ARIA_GCM_Decrypt, **1367**
 CRYPTO_ARIA_GCM_Encrypt, **1366**
 CRYPTO_ARIA_InitDecrypt, **1352**
 CRYPTO_ARIA_InitEncrypt, **1351**
 CRYPTO_ARIA_Install, **1348**, 2653, 2658, 2662, 2665
 CRYPTO_ARIA_IsInstalled, **1349**
 CRYPTO_ARIA_Kill, **1353**
 CRYPTO_ARIA_OFB_Decrypt, **1363**
 CRYPTO_ARIA_OFB_Encrypt, **1362**
 CRYPTO_ARIA_QueryInstall, **1350**
 CRYPTO_ARIA_RFC5794_SelfTest, **1396**
 CRYPTO_BLAKE2B_Add, **51**
 CRYPTO_BLAKE2B_Calc, **52**
 CRYPTO_BLAKE2B_Calc_512, **53**
 CRYPTO_BLAKE2B_Final, **54**
 CRYPTO_BLAKE2B_Final_512, **55**
 CRYPTO_BLAKE2B_Get, **56**
 CRYPTO_BLAKE2B_Init, **57**
 CRYPTO_BLAKE2B_Install, **58**
 CRYPTO_BLAKE2B_IsInstalled, **59**
 CRYPTO_BLAKE2B_Kill, **60**
 CRYPTO_BLAKE2B_QueryInstall, **61**
 CRYPTO_BLAKE2B_Ref_SelfTest, **70**
 CRYPTO_BLAKE2B_RFC7693_SelfTest, **69**
 CRYPTO_BLAKE2S_Add, **73**
 CRYPTO_BLAKE2S_Calc, **74**
 CRYPTO_BLAKE2S_Calc_256, **75**
 CRYPTO_BLAKE2S_Final, **76**

CRYPTO_BLAKE2S_Final_256, **77**
 CRYPTO_BLAKE2S_Get, **78**
 CRYPTO_BLAKE2S_Init, **79**
 CRYPTO_BLAKE2S_Install, **80**
 CRYPTO_BLAKE2S_IsInstalled, **81**
 CRYPTO_BLAKE2S_Kill, **82**
 CRYPTO_BLAKE2S_QueryInstall, **83**
 CRYPTO_BLAKE2S_RFC7693_SelfTest, **91**
 CRYPTO_BLOWFISH_CBC_Decrypt, **1521**
 CRYPTO_BLOWFISH_CBC_Encrypt, **1520**
 CRYPTO_BLOWFISH_CTR_Decrypt, **1525**
 CRYPTO_BLOWFISH_CTR_Encrypt, **1524**
 CRYPTO_BLOWFISH_Decrypt, **1517**
 CRYPTO_BLOWFISH_ECB_Decrypt, **1519**
 CRYPTO_BLOWFISH_ECB_Encrypt, **1518**
 CRYPTO_BLOWFISH_Encrypt, **1516**
 CRYPTO_BLOWFISH_InitDecrypt, **1514**
 CRYPTO_BLOWFISH_InitEncrypt, **1513**
 CRYPTO_BLOWFISH_Install, **1510**, 2653, 2658
 CRYPTO_BLOWFISH_IsInstalled, **1511**
 CRYPTO_BLOWFISH_Kill, **1515**
 CRYPTO_BLOWFISH_OFB_Decrypt, **1523**
 CRYPTO_BLOWFISH_OFB_Encrypt, **1522**
 CRYPTO_BLOWFISH_QueryInstall, **1512**
 CRYPTO_BLOWFISH_Schneier_SelfTest, **1551**
 CRYPTO_BUFFER_Copy, **2669**
 CRYPTO_BUFFER_Cursor, **2670**
 CRYPTO_BUFFER_CursorDistance, **2671**
 CRYPTO_BUFFER_CursorIndex, **2672**
 CRYPTO_BUFFER_Data, **2673**
 CRYPTO_BUFFER_Init, **2674**
 CRYPTO_BUFFER_Insert, **2675**
 CRYPTO_BUFFER_Mark, **2679**
 CRYPTO_BUFFER_MarkU16, **2681**
 CRYPTO_BUFFER_MarkU24, **2682**
 CRYPTO_BUFFER_MarkU32, **2683**
 CRYPTO_BUFFER_MarkU8, **2680**
 CRYPTO_BUFFER_MPI_WrCounted, **2676**
 CRYPTO_BUFFER_MPI_WrRaw, **2677**
 CRYPTO_BUFFER_MPI_WrRawLE, **2678**
 CRYPTO_BUFFER_Overflow, **2684**
 CRYPTO_BUFFER_PatchU16BE, **2686**
 CRYPTO_BUFFER_PatchU16LE, **2689**
 CRYPTO_BUFFER_PatchU24BE, **2687**
 CRYPTO_BUFFER_PatchU24LE, **2690**
 CRYPTO_BUFFER_PatchU32BE, **2688**
 CRYPTO_BUFFER_PatchU32LE, **2691**
 CRYPTO_BUFFER_PatchU8, **2685**
 CRYPTO_BUFFER_Reserve, **2692**
 CRYPTO_BUFFER_Reset, **2693**
 CRYPTO_BUFFER_SetCursor, **2694**
 CRYPTO_BUFFER_SetCursorIndex, **2695**
 CRYPTO_BUFFER_Skip, **2696**
 CRYPTO_BUFFER_SpaceLeft, **2697**
 CRYPTO_BUFFER_Status, **2698**
 CRYPTO_BUFFER_Wr, **2699**
 CRYPTO_BUFFER_WrCntU32BE, **2700**
 CRYPTO_BUFFER_WrLogical, **2701**
 CRYPTO_BUFFER_WrMultiU8, **2702**
 CRYPTO_BUFFER_WrStr, **2703**
 CRYPTO_BUFFER_WrStrLn, **2704**
 CRYPTO_BUFFER_WrU16BE, **2705**
 CRYPTO_BUFFER_WrU16LE, **2706**
 CRYPTO_BUFFER_WrU24BE, **2707**
 CRYPTO_BUFFER_WrU24LE, **2708**
 CRYPTO_BUFFER_WrU32BE, **2709**
 CRYPTO_BUFFER_WrU32LE, **2710**
 CRYPTO_BUFFER_WrU64BE, **2711**
 CRYPTO_BUFFER_WrU64LE, **2712**
 CRYPTO_BUFFER_WrU8, **2713**
 CRYPTO_CAMELLIA_CBC_Decrypt, **1411**
 CRYPTO_CAMELLIA_CBC_Encrypt, **1410**
 CRYPTO_CAMELLIA_CCM_Decrypt, **1417**, 2767
 CRYPTO_CAMELLIA_CCM_Encrypt, **1416**, 2767
 CRYPTO_CAMELLIA_CTR_Decrypt, **1415**
 CRYPTO_CAMELLIA_CTR_Encrypt, **1414**

CRYPTO_CAMELLIA_Decrypt, **1407**
 CRYPTO_CAMELLIA_ECB_Decrypt, **1409**
 CRYPTO_CAMELLIA_ECB_Encrypt, **1408**
 CRYPTO_CAMELLIA_Encrypt, **1406**
 CRYPTO_CAMELLIA_GCM_Decrypt, **1419**, 2767
 CRYPTO_CAMELLIA_GCM_Encrypt, **1418**, 2766
 CRYPTO_CAMELLIA_InitDecrypt, **1404**
 CRYPTO_CAMELLIA_InitEncrypt, **1403**
 CRYPTO_CAMELLIA_Install, **1400**, 2653, 2658, 2662, 2665
 CRYPTO_CAMELLIA_IsInstalled, **1401**
 CRYPTO_CAMELLIA_Kill, **1405**
 CRYPTO_CAMELLIA_NTT_SelfTest, **1448**
 CRYPTO_CAMELLIA_OFB_Decrypt, **1413**
 CRYPTO_CAMELLIA_OFB_Encrypt, **1412**
 CRYPTO_CAMELLIA_QueryInstall, **1402**, 2767
 CRYPTO_CAMELLIA_RFC5528_SelfTest, **1449**
 CRYPTO_CAST_CBC_Decrypt, **1463**
 CRYPTO_CAST_CBC_Encrypt, **1464**
 CRYPTO_CAST_CTR_Decrypt, **1467**
 CRYPTO_CAST_CTR_Encrypt, **1468**
 CRYPTO_CAST_Decrypt, **1460**
 CRYPTO_CAST_ECB_Decrypt, **1461**
 CRYPTO_CAST_ECB_Encrypt, **1462**
 CRYPTO_CAST_Encrypt, **1459**
 CRYPTO_CAST_InitDecrypt, **1457**
 CRYPTO_CAST_InitEncrypt, **1456**
 CRYPTO_CAST_Install, **1453**, 2653, 2658
 CRYPTO_CAST_IsInstalled, **1454**
 CRYPTO_CAST_Kill, **1458**
 CRYPTO_CAST_OFB_Decrypt, **1465**
 CRYPTO_CAST_OFB_Encrypt, **1466**
 CRYPTO_CAST_QueryInstall, **1455**
 CRYPTO_CAST_RFC2144_SelfTest, **1493**
 CRYPTO_CHACHA20_InitDecrypt_32_96, **1497**
 CRYPTO_CHACHA20_InitDecrypt_64_64, **1499**
 CRYPTO_CHACHA20_InitEncrypt_32_96, **1496**
 CRYPTO_CHACHA20_InitEncrypt_64_64, **1498**
 CRYPTO_CHACHA20_Kill, **1502**
 CRYPTO_CHACHA20_POLY1305_Decrypt, **1505**
 CRYPTO_CHACHA20_POLY1305_Encrypt, **1504**
 CRYPTO_CHACHA20_POLY1305_GenKey, **1503**
 CRYPTO_CHACHA20_RFC7539_SelfTest, **1507**
 CRYPTO_CHACHA20_SetIV, **1501**
 CRYPTO_CHACHA20_SetPos, **1500**
 CRYPTO_CIPHER_AES_128_InitDecrypt, **1220**
 CRYPTO_CIPHER_AES_128_InitEncrypt, **19**, **1216**
 CRYPTO_CIPHER_AES_192_InitDecrypt, **1221**
 CRYPTO_CIPHER_AES_192_InitEncrypt, **1217**
 CRYPTO_CIPHER_AES_256_InitDecrypt, **1222**
 CRYPTO_CIPHER_AES_256_InitEncrypt, **1218**
 CRYPTO_CIPHER_AES_CBC_Decrypt, **1228**
 CRYPTO_CIPHER_AES_CBC_Encrypt, **1227**
 CRYPTO_CIPHER_AES_CCM_Decrypt, **1236**
 CRYPTO_CIPHER_AES_CCM_Encrypt, **1235**
 CRYPTO_CIPHER_AES_CTR_0_16_Decrypt, **1233**
 CRYPTO_CIPHER_AES_CTR_0_16_Encrypt, **1230**
 CRYPTO_CIPHER_AES_CTR_12_4_Decrypt, **1234**
 CRYPTO_CIPHER_AES_CTR_12_4_Encrypt, **1231**
 CRYPTO_CIPHER_AES_CTR_Decrypt, **1232**
 CRYPTO_CIPHER_AES_CTR_Encrypt, **1229**
 CRYPTO_CIPHER_AES_Decrypt, **1224**
 CRYPTO_CIPHER_AES_ECB_Decrypt, **1226**
 CRYPTO_CIPHER_AES_ECB_Encrypt, **1225**
 CRYPTO_CIPHER_AES_Encrypt, **1223**
 CRYPTO_CIPHER_AES_GCM_Decrypt, **1238**
 CRYPTO_CIPHER_AES_GCM_Encrypt, **1237**
 CRYPTO_CIPHER_AES_InitDecrypt, **1219**
 CRYPTO_CIPHER_AES_InitEncrypt, **18**, **18**, **1215**
 CRYPTO_CIPHER_ARIA_128_InitDecrypt, **1374**
 CRYPTO_CIPHER_ARIA_128_InitEncrypt, **1370**
 CRYPTO_CIPHER_ARIA_192_InitDecrypt, **1371**
 CRYPTO_CIPHER_ARIA_192_InitEncrypt, **1375**
 CRYPTO_CIPHER_ARIA_256_InitDecrypt, **1376**
 CRYPTO_CIPHER_ARIA_256_InitEncrypt, **1372**
 CRYPTO_CIPHER_ARIA_CBC_Decrypt, **1382**
 CRYPTO_CIPHER_ARIA_CBC_Encrypt, **1381**

CRYPTO_CIPHER_ARIA_CCM_Decrypt, **1392**
 CRYPTO_CIPHER_ARIA_CCM_Encrypt, **1391**
 CRYPTO_CIPHER_ARIA_CTR_0_16_Decrypt, **1389**
 CRYPTO_CIPHER_ARIA_CTR_0_16_Encrypt, **1386**
 CRYPTO_CIPHER_ARIA_CTR_12_4_Decrypt, **1390**
 CRYPTO_CIPHER_ARIA_CTR_12_4_Encrypt, **1387**
 CRYPTO_CIPHER_ARIA_CTR_Decrypt, **1388**
 CRYPTO_CIPHER_ARIA_CTR_Encrypt, **1385**
 CRYPTO_CIPHER_ARIA_Decrypt, **1378**
 CRYPTO_CIPHER_ARIA_ECB_Decrypt, **1380**
 CRYPTO_CIPHER_ARIA_ECB_Encrypt, **1379**
 CRYPTO_CIPHER_ARIA_Encrypt, **1377**
 CRYPTO_CIPHER_ARIA_GCM_Decrypt, **1394**
 CRYPTO_CIPHER_ARIA_GCM_Encrypt, **1393**
 CRYPTO_CIPHER_ARIA_InitDecrypt, **1373**
 CRYPTO_CIPHER_ARIA_InitEncrypt, **1369**
 CRYPTO_CIPHER_ARIA_OFB_Decrypt, **1384**
 CRYPTO_CIPHER_ARIA_OFB_Encrypt, **1383**
 CRYPTO_CIPHER_BLOWFISH_128_InitDecrypt, **1533**
 CRYPTO_CIPHER_BLOWFISH_128_InitEncrypt, **1529**
 CRYPTO_CIPHER_BLOWFISH_192_InitDecrypt, **1534**
 CRYPTO_CIPHER_BLOWFISH_192_InitEncrypt, **1530**
 CRYPTO_CIPHER_BLOWFISH_256_InitDecrypt, **1535**
 CRYPTO_CIPHER_BLOWFISH_256_InitEncrypt, **1531**
 CRYPTO_CIPHER_BLOWFISH_CBC_Decrypt, **1541**
 CRYPTO_CIPHER_BLOWFISH_CBC_Encrypt, **1540**
 CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Decrypt, **1548**
 CRYPTO_CIPHER_BLOWFISH_CTR_0_8_Encrypt, **1545**
 CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Decrypt, **1549**
 CRYPTO_CIPHER_BLOWFISH_CTR_4_4_Encrypt, **1546**
 CRYPTO_CIPHER_BLOWFISH_CTR_Decrypt, **1547**
 CRYPTO_CIPHER_BLOWFISH_CTR_Encrypt, **1544**
 CRYPTO_CIPHER_BLOWFISH_Decrypt, **1537**
 CRYPTO_CIPHER_BLOWFISH_ECB_Decrypt, **1539**
 CRYPTO_CIPHER_BLOWFISH_ECB_Encrypt, **1538**
 CRYPTO_CIPHER_BLOWFISH_Encrypt, **1536**
 CRYPTO_CIPHER_BLOWFISH_InitDecrypt, **1532**
 CRYPTO_CIPHER_BLOWFISH_InitEncrypt, **1528**
 CRYPTO_CIPHER_BLOWFISH_OFB_Decrypt, **1543**
 CRYPTO_CIPHER_BLOWFISH_OFB_Encrypt, **1542**
 CRYPTO_CIPHER_CAMELLIA_128_InitDecrypt, **1426**
 CRYPTO_CIPHER_CAMELLIA_128_InitEncrypt, **1422**
 CRYPTO_CIPHER_CAMELLIA_192_InitDecrypt, **1427**
 CRYPTO_CIPHER_CAMELLIA_192_InitEncrypt, **1423**
 CRYPTO_CIPHER_CAMELLIA_256_InitDecrypt, **1428**
 CRYPTO_CIPHER_CAMELLIA_256_InitEncrypt, **1424**
 CRYPTO_CIPHER_CAMELLIA_CBC_Decrypt, **1434**
 CRYPTO_CIPHER_CAMELLIA_CBC_Encrypt, **1433**
 CRYPTO_CIPHER_CAMELLIA_CCM_Decrypt, **1444**
 CRYPTO_CIPHER_CAMELLIA_CCM_Encrypt, **1443**
 CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Decrypt, **1441**
 CRYPTO_CIPHER_CAMELLIA_CTR_0_16_Encrypt, **1438**
 CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Decrypt, **1442**
 CRYPTO_CIPHER_CAMELLIA_CTR_12_4_Encrypt, **1439**
 CRYPTO_CIPHER_CAMELLIA_CTR_Decrypt, **1440**
 CRYPTO_CIPHER_CAMELLIA_CTR_Encrypt, **1437**
 CRYPTO_CIPHER_CAMELLIA_Decrypt, **1430**
 CRYPTO_CIPHER_CAMELLIA_ECB_Decrypt, **1432**
 CRYPTO_CIPHER_CAMELLIA_ECB_Encrypt, **1431**
 CRYPTO_CIPHER_CAMELLIA_Encrypt, **1429**
 CRYPTO_CIPHER_CAMELLIA_GCM_Decrypt, **1446**
 CRYPTO_CIPHER_CAMELLIA_GCM_Encrypt, **1445**
 CRYPTO_CIPHER_CAMELLIA_InitDecrypt, **1425**
 CRYPTO_CIPHER_CAMELLIA_InitEncrypt, **1421**
 CRYPTO_CIPHER_CAMELLIA_OFB_Decrypt, **1436**
 CRYPTO_CIPHER_CAMELLIA_OFB_Encrypt, **1435**
 CRYPTO_CIPHER_CAST_128_InitDecrypt, **1475**
 CRYPTO_CIPHER_CAST_128_InitEncrypt, **1471**
 CRYPTO_CIPHER_CAST_192_InitDecrypt, **1476**
 CRYPTO_CIPHER_CAST_192_InitEncrypt, **1472**
 CRYPTO_CIPHER_CAST_256_InitDecrypt, **1477**
 CRYPTO_CIPHER_CAST_256_InitEncrypt, **1473**
 CRYPTO_CIPHER_CAST_CBC_Decrypt, **1483**
 CRYPTO_CIPHER_CAST_CBC_Encrypt, **1482**
 CRYPTO_CIPHER_CAST_CTR_0_8_Decrypt, **1490**
 CRYPTO_CIPHER_CAST_CTR_0_8_Encrypt, **1487**

CRYPTO_CIPHER_CAST_CTR_4_4_Decrypt, **1491**
 CRYPTO_CIPHER_CAST_CTR_4_4_Encrypt, **1488**
 CRYPTO_CIPHER_CAST_CTR_Decrypt, **1489**
 CRYPTO_CIPHER_CAST_CTR_Encrypt, **1486**
 CRYPTO_CIPHER_CAST_Decrypt, **1479**
 CRYPTO_CIPHER_CAST_ECB_Decrypt, **1481**
 CRYPTO_CIPHER_CAST_ECB_Encrypt, **1480**
 CRYPTO_CIPHER_CAST_Encrypt, **1478**
 CRYPTO_CIPHER_CAST_InitDecrypt, **1474**
 CRYPTO_CIPHER_CAST_InitEncrypt, **1470**
 CRYPTO_CIPHER_CAST_OFB_Decrypt, **1485**
 CRYPTO_CIPHER_CAST_OFB_Encrypt, **1484**
 CRYPTO_CIPHER_CBC_Decrypt, **1655**, 2756, 2762, 2766
 CRYPTO_CIPHER_CBC_Encrypt, **1654**, 2756, 2762, 2766
 CRYPTO_CIPHER_CCM_Cipher, **1660**
 CRYPTO_CIPHER_CTR_Decrypt, **1659**
 CRYPTO_CIPHER_CTR_Encrypt, **1658**
 CRYPTO_CIPHER_ECB_Decrypt, **1653**, 2756, 2765
 CRYPTO_CIPHER_ECB_Encrypt, **1652**, 2755, 2765
 CRYPTO_CIPHER_GCM_Cipher, **1661**
 CRYPTO_CIPHER_GCM_GF128_Multiply, **1662**
 CRYPTO_CIPHER_GCM_Plain_Cipher, **1663**
 CRYPTO_CIPHER_GCM_Shoup_8b_Cipher, **1664**
 CRYPTO_CIPHER_IDEA_128_InitDecrypt, **1276**
 CRYPTO_CIPHER_IDEA_128_InitEncrypt, **1274**
 CRYPTO_CIPHER_IDEA_CBC_Decrypt, **1282**
 CRYPTO_CIPHER_IDEA_CBC_Encrypt, **1281**
 CRYPTO_CIPHER_IDEA_CTR_0_8_Decrypt, **1289**
 CRYPTO_CIPHER_IDEA_CTR_0_8_Encrypt, **1286**
 CRYPTO_CIPHER_IDEA_CTR_4_4_Decrypt, **1290**
 CRYPTO_CIPHER_IDEA_CTR_4_4_Encrypt, **1287**
 CRYPTO_CIPHER_IDEA_CTR_Decrypt, **1288**
 CRYPTO_CIPHER_IDEA_CTR_Encrypt, **1285**
 CRYPTO_CIPHER_IDEA_Decrypt, **1278**
 CRYPTO_CIPHER_IDEA_ECB_Decrypt, **1280**
 CRYPTO_CIPHER_IDEA_ECB_Encrypt, **1279**
 CRYPTO_CIPHER_IDEA_Encrypt, **1277**
 CRYPTO_CIPHER_IDEA_InitDecrypt, **1275**
 CRYPTO_CIPHER_IDEA_InitEncrypt, **1273**
 CRYPTO_CIPHER_IDEA_OFB_Decrypt, **1284**
 CRYPTO_CIPHER_IDEA_OFB_Encrypt, **1283**
 CRYPTO_CIPHER_OFB_Decrypt, **1657**
 CRYPTO_CIPHER_OFB_Encrypt, **1656**
 CRYPTO_CIPHER_PRESENT_128_InitDecrypt, **1629**
 CRYPTO_CIPHER_PRESENT_128_InitEncrypt, **1626**
 CRYPTO_CIPHER_PRESENT_80_InitDecrypt, **1628**
 CRYPTO_CIPHER_PRESENT_80_InitEncrypt, **1625**
 CRYPTO_CIPHER_PRESENT_CBC_Decrypt, **1635**
 CRYPTO_CIPHER_PRESENT_CBC_Encrypt, **1634**
 CRYPTO_CIPHER_PRESENT_CTR_0_8_Decrypt, **1642**
 CRYPTO_CIPHER_PRESENT_CTR_0_8_Encrypt, **1639**
 CRYPTO_CIPHER_PRESENT_CTR_4_4_Decrypt, **1643**
 CRYPTO_CIPHER_PRESENT_CTR_4_4_Encrypt, **1640**
 CRYPTO_CIPHER_PRESENT_CTR_Decrypt, **1641**
 CRYPTO_CIPHER_PRESENT_CTR_Encrypt, **1638**
 CRYPTO_CIPHER_PRESENT_Decrypt, **1631**
 CRYPTO_CIPHER_PRESENT_ECB_Decrypt, **1633**
 CRYPTO_CIPHER_PRESENT_ECB_Encrypt, **1632**
 CRYPTO_CIPHER_PRESENT_Encrypt, **1630**
 CRYPTO_CIPHER_PRESENT_InitDecrypt, **1627**
 CRYPTO_CIPHER_PRESENT_InitEncrypt, **1624**
 CRYPTO_CIPHER_PRESENT_OFB_Decrypt, **1637**
 CRYPTO_CIPHER_PRESENT_OFB_Encrypt, **1636**
 CRYPTO_CIPHER_SEED_128_InitDecrypt, **1322**
 CRYPTO_CIPHER_SEED_128_InitEncrypt, **1318**
 CRYPTO_CIPHER_SEED_192_InitDecrypt, **1319**
 CRYPTO_CIPHER_SEED_192_InitEncrypt, **1323**
 CRYPTO_CIPHER_SEED_256_InitDecrypt, **1324**
 CRYPTO_CIPHER_SEED_256_InitEncrypt, **1320**
 CRYPTO_CIPHER_SEED_CBC_Decrypt, **1330**
 CRYPTO_CIPHER_SEED_CBC_Encrypt, **1329**
 CRYPTO_CIPHER_SEED_CCM_Decrypt, **1340**
 CRYPTO_CIPHER_SEED_CCM_Encrypt, **1339**
 CRYPTO_CIPHER_SEED_CTR_0_16_Decrypt, **1337**
 CRYPTO_CIPHER_SEED_CTR_0_16_Encrypt, **1334**
 CRYPTO_CIPHER_SEED_CTR_12_4_Decrypt, **1338**

CRYPTO_CIPHER_SEED_CTR_12_4_Encrypt, **1335**
 CRYPTO_CIPHER_SEED_CTR_Decrypt, **1336**
 CRYPTO_CIPHER_SEED_CTR_Encrypt, **1333**
 CRYPTO_CIPHER_SEED_Decrypt, **1326**
 CRYPTO_CIPHER_SEED_ECB_Decrypt, **1328**
 CRYPTO_CIPHER_SEED_ECB_Encrypt, **1327**
 CRYPTO_CIPHER_SEED_Encrypt, **1325**
 CRYPTO_CIPHER_SEED_GCM_Decrypt, **1342**
 CRYPTO_CIPHER_SEED_GCM_Encrypt, **1341**
 CRYPTO_CIPHER_SEED_InitDecrypt, **1321**
 CRYPTO_CIPHER_SEED_InitEncrypt, **1317**
 CRYPTO_CIPHER_SEED_OFB_Decrypt, **1332**
 CRYPTO_CIPHER_SEED_OFB_Encrypt, **1331**
 CRYPTO_CIPHER_TDES_128_InitDecrypt, **1171**
 CRYPTO_CIPHER_TDES_128_InitEncrypt, **1167**
 CRYPTO_CIPHER_TDES_192_InitDecrypt, **1172**
 CRYPTO_CIPHER_TDES_192_InitEncrypt, **1168**
 CRYPTO_CIPHER_TDES_64_InitDecrypt, **1170**
 CRYPTO_CIPHER_TDES_64_InitEncrypt, **1166**
 CRYPTO_CIPHER_TDES_CBC_Decrypt, **1178**
 CRYPTO_CIPHER_TDES_CBC_Encrypt, **1177**
 CRYPTO_CIPHER_TDES_CTR_0_8_Decrypt, **1185**
 CRYPTO_CIPHER_TDES_CTR_0_8_Encrypt, **1182**
 CRYPTO_CIPHER_TDES_CTR_4_4_Decrypt, **1186**
 CRYPTO_CIPHER_TDES_CTR_4_4_Encrypt, **1183**
 CRYPTO_CIPHER_TDES_CTR_Decrypt, **1184**
 CRYPTO_CIPHER_TDES_CTR_Encrypt, **1181**
 CRYPTO_CIPHER_TDES_Decrypt, **1174**
 CRYPTO_CIPHER_TDES_ECB_Decrypt, **1176**
 CRYPTO_CIPHER_TDES_ECB_Encrypt, **1175**
 CRYPTO_CIPHER_TDES_Encrypt, **1173**
 CRYPTO_CIPHER_TDES_InitDecrypt, **1169**
 CRYPTO_CIPHER_TDES_InitEncrypt, **1165**
 CRYPTO_CIPHER_TDES_OFB_Decrypt, **1180**
 CRYPTO_CIPHER_TDES_OFB_Encrypt, **1179**
 CRYPTO_CIPHER_TWOFISH_128_InitDecrypt, **1581**
 CRYPTO_CIPHER_TWOFISH_128_InitEncrypt, **1577**
 CRYPTO_CIPHER_TWOFISH_192_InitDecrypt, **1582**
 CRYPTO_CIPHER_TWOFISH_192_InitEncrypt, **1578**
 CRYPTO_CIPHER_TWOFISH_256_InitDecrypt, **1583**
 CRYPTO_CIPHER_TWOFISH_256_InitEncrypt, **1579**
 CRYPTO_CIPHER_TWOFISH_CBC_Decrypt, **1589**
 CRYPTO_CIPHER_TWOFISH_CBC_Encrypt, **1588**
 CRYPTO_CIPHER_TWOFISH_CCM_Decrypt, **1599**
 CRYPTO_CIPHER_TWOFISH_CCM_Encrypt, **1598**
 CRYPTO_CIPHER_TWOFISH_CTR_0_16_Decrypt, **1596**
 CRYPTO_CIPHER_TWOFISH_CTR_0_16_Encrypt, **1593**
 CRYPTO_CIPHER_TWOFISH_CTR_12_4_Decrypt, **1597**
 CRYPTO_CIPHER_TWOFISH_CTR_12_4_Encrypt, **1594**
 CRYPTO_CIPHER_TWOFISH_CTR_Decrypt, **1595**
 CRYPTO_CIPHER_TWOFISH_CTR_Encrypt, **1592**
 CRYPTO_CIPHER_TWOFISH_Decrypt, **1585**
 CRYPTO_CIPHER_TWOFISH_ECB_Decrypt, **1587**
 CRYPTO_CIPHER_TWOFISH_ECB_Encrypt, **1586**
 CRYPTO_CIPHER_TWOFISH_Encrypt, **1584**
 CRYPTO_CIPHER_TWOFISH_GCM_Decrypt, **1601**
 CRYPTO_CIPHER_TWOFISH_GCM_Encrypt, **1600**
 CRYPTO_CIPHER_TWOFISH_InitDecrypt, **1580**
 CRYPTO_CIPHER_TWOFISH_InitEncrypt, **1576**
 CRYPTO_CIPHER_TWOFISH_OFB_Decrypt, **1591**
 CRYPTO_CIPHER_TWOFISH_OFB_Encrypt, **1590**
 CRYPTO_CMAC_AES_Add, **414**
 CRYPTO_CMAC_AES_Calc, **415**
 CRYPTO_CMAC_AES_Calc_128, **416**
 CRYPTO_CMAC_AES_CAVS_SelfTest, **430**
 CRYPTO_CMAC_AES_Final, **417**
 CRYPTO_CMAC_AES_Final_128, **418**
 CRYPTO_CMAC_AES_Init, **419**
 CRYPTO_CMAC_AES_InitEx, **420**
 CRYPTO_CMAC_AES_Kill, **421**
 CRYPTO_CMAC_ARIA_Add, **502**
 CRYPTO_CMAC_ARIA_Calc, **503**
 CRYPTO_CMAC_ARIA_Calc_128, **504**
 CRYPTO_CMAC_ARIA_Final, **505**
 CRYPTO_CMAC_ARIA_Final_128, **506**
 CRYPTO_CMAC_ARIA_Init, **507**

CRYPTO_CMAC_ARIA_InitEx, **508**
 CRYPTO_CMAC_ARIA_Kill, **509**
 CRYPTO_CMAC_BLOWFISH_Add, **552**
 CRYPTO_CMAC_BLOWFISH_Calc, **553**
 CRYPTO_CMAC_BLOWFISH_Calc_64, **554**
 CRYPTO_CMAC_BLOWFISH_Final, **555**
 CRYPTO_CMAC_BLOWFISH_Final_64, **556**
 CRYPTO_CMAC_BLOWFISH_Init, **557**
 CRYPTO_CMAC_BLOWFISH_InitEx, **558**
 CRYPTO_CMAC_BLOWFISH_Kill, **559**
 CRYPTO_CMAC_CAMELLIA_Add, **519**
 CRYPTO_CMAC_CAMELLIA_Calc, **520**
 CRYPTO_CMAC_CAMELLIA_Calc_128, **521**
 CRYPTO_CMAC_CAMELLIA_Final, **522**
 CRYPTO_CMAC_CAMELLIA_Final_128, **523**
 CRYPTO_CMAC_CAMELLIA_Init, **524**
 CRYPTO_CMAC_CAMELLIA_InitEx, **525**
 CRYPTO_CMAC_CAMELLIA_Kill, **526**
 CRYPTO_CMAC_CAST_Add, **468**
 CRYPTO_CMAC_CAST_Calc, **469**
 CRYPTO_CMAC_CAST_Calc_64, **470**
 CRYPTO_CMAC_CAST_Final, **471**
 CRYPTO_CMAC_CAST_Final_64, **472**
 CRYPTO_CMAC_CAST_Init, **473**
 CRYPTO_CMAC_CAST_InitEx, **474**
 CRYPTO_CMAC_CAST_Kill, **475**
 CRYPTO_CMAC_IDEA_Add, **452**
 CRYPTO_CMAC_IDEA_Calc, **453**
 CRYPTO_CMAC_IDEA_Calc_64, **454**
 CRYPTO_CMAC_IDEA_Final, **455**
 CRYPTO_CMAC_IDEA_Final_64, **456**
 CRYPTO_CMAC_IDEA_Init, **457**
 CRYPTO_CMAC_IDEA_InitEx, **458**
 CRYPTO_CMAC_IDEA_Kill, **459**
 CRYPTO_CMAC_PRESENT_Add, **569**
 CRYPTO_CMAC_PRESENT_Calc, **570**
 CRYPTO_CMAC_PRESENT_Calc_64, **571**
 CRYPTO_CMAC_PRESENT_Final, **572**
 CRYPTO_CMAC_PRESENT_Final_64, **573**
 CRYPTO_CMAC_PRESENT_Init, **574**
 CRYPTO_CMAC_PRESENT_InitEx, **575**
 CRYPTO_CMAC_PRESENT_Kill, **576**
 CRYPTO_CMAC_SEED_Add, **485**
 CRYPTO_CMAC_SEED_Calc, **486**
 CRYPTO_CMAC_SEED_Calc_128, **487**
 CRYPTO_CMAC_SEED_Final, **488**
 CRYPTO_CMAC_SEED_Final_128, **489**
 CRYPTO_CMAC_SEED_Init, **490**
 CRYPTO_CMAC_SEED_InitEx, **491**
 CRYPTO_CMAC_SEED_Kill, **492**
 CRYPTO_CMAC_TDES_Add, **433**
 CRYPTO_CMAC_TDES_Calc, **434**
 CRYPTO_CMAC_TDES_Calc_64, **435**
 CRYPTO_CMAC_TDES_CAVS_SelfTest, **449**
 CRYPTO_CMAC_TDES_Final, **436**
 CRYPTO_CMAC_TDES_Final_64, **437**
 CRYPTO_CMAC_TDES_Init, **438**
 CRYPTO_CMAC_TDES_InitEx, **439**
 CRYPTO_CMAC_TDES_Kill, **440**
 CRYPTO_CMAC_TWOFISH_Add, **536**
 CRYPTO_CMAC_TWOFISH_Calc, **537**
 CRYPTO_CMAC_TWOFISH_Calc_128, **538**
 CRYPTO_CMAC_TWOFISH_Final, **539**
 CRYPTO_CMAC_TWOFISH_Final_128, **540**
 CRYPTO_CMAC_TWOFISH_Init, **541**
 CRYPTO_CMAC_TWOFISH_InitEx, **542**
 CRYPTO_CMAC_TWOFISH_Kill, **543**
 CRYPTO_CMS_Rd_CCMParameters, **2634**
 CRYPTO_CMS_Rd_GCMParameters, **2635**
 CRYPTO_CSHAKE128_Calc, **1945**
 CRYPTO_CSHAKE256_Calc, **1946**
 CRYPTO_CSHAKE_Add, **1948**
 CRYPTO_CSHAKE_BlockPad, **1952**
 CRYPTO_CSHAKE_EncodeStr, **1951**
 CRYPTO_CSHAKE_Get, **1953**
 CRYPTO_CSHAKE_Init, **1947**

CRYPTO_CSHAKE_Kill, **1954**
 CRYPTO_CSHAKE_LeftEncode, **1949**
 CRYPTO_CSHAKE_RightEncode, **1950**
 CRYPTO_DH_KA_Agree, **2374**
 CRYPTO_DH_KA_GenKeys, **2376**
 CRYPTO_DH_KA_Init, **2372**
 CRYPTO_DH_KA_Kill, **2375**
 CRYPTO_DH_KA_SEGGER_SelfTest, **2378**
 CRYPTO_DH_KA_Start, **2373**
 CRYPTO_DRBG_CTR_AES128_CAVS_SelfTest, **1787**
 CRYPTO_DRBG_CTR_AES128_Get, **1785**
 CRYPTO_DRBG_CTR_AES128_Init, **1783**
 CRYPTO_DRBG_CTR_AES128_Reseed, **1784**
 CRYPTO_DRBG_CTR_AES192_CAVS_SelfTest, **1793**
 CRYPTO_DRBG_CTR_AES192_Get, **1791**
 CRYPTO_DRBG_CTR_AES192_Init, **1789**
 CRYPTO_DRBG_CTR_AES192_Reseed, **1790**
 CRYPTO_DRBG_CTR_AES256_CAVS_SelfTest, **1799**
 CRYPTO_DRBG_CTR_AES256_Get, **1797**
 CRYPTO_DRBG_CTR_AES256_Init, **1795**
 CRYPTO_DRBG_CTR_AES256_Reseed, **1796**
 CRYPTO_DRBG_CTR_TDES_CAVS_SelfTest, **1781**
 CRYPTO_DRBG_CTR_TDES_Get, **1779**
 CRYPTO_DRBG_CTR_TDES_Init, **1777**
 CRYPTO_DRBG_CTR_TDES_Reseed, **1778**
 CRYPTO_DRBG_HASH_SHA1_CAVS_SelfTest, **1697**
 CRYPTO_DRBG_HASH_SHA1_Get, **1695**
 CRYPTO_DRBG_HASH_SHA1_Init, **1693**
 CRYPTO_DRBG_HASH_SHA1_Reseed, **1694**
 CRYPTO_DRBG_HASH_SHA224_CAVS_SelfTest, **1703**
 CRYPTO_DRBG_HASH_SHA224_Get, **1701**
 CRYPTO_DRBG_HASH_SHA224_Init, **1699**
 CRYPTO_DRBG_HASH_SHA224_Reseed, **1700**
 CRYPTO_DRBG_HASH_SHA256_CAVS_SelfTest, **1709**
 CRYPTO_DRBG_HASH_SHA256_Get, **1707**
 CRYPTO_DRBG_HASH_SHA256_Init, **1705**
 CRYPTO_DRBG_HASH_SHA256_Reseed, **1706**
 CRYPTO_DRBG_HASH_SHA384_CAVS_SelfTest, **1715**
 CRYPTO_DRBG_HASH_SHA384_Get, **1713**
 CRYPTO_DRBG_HASH_SHA384_Init, **1711**
 CRYPTO_DRBG_HASH_SHA384_Reseed, **1712**
 CRYPTO_DRBG_HASH_SHA512_224_CAVS_SelfTest, **1727**
 CRYPTO_DRBG_HASH_SHA512_224_Get, **1725**
 CRYPTO_DRBG_HASH_SHA512_224_Init, **1723**
 CRYPTO_DRBG_HASH_SHA512_224_Reseed, **1724**
 CRYPTO_DRBG_HASH_SHA512_256_CAVS_SelfTest, **1733**
 CRYPTO_DRBG_HASH_SHA512_256_Get, **1731**
 CRYPTO_DRBG_HASH_SHA512_256_Init, **1729**
 CRYPTO_DRBG_HASH_SHA512_256_Reseed, **1730**
 CRYPTO_DRBG_HASH_SHA512_CAVS_SelfTest, **1721**
 CRYPTO_DRBG_HASH_SHA512_Get, **1719**
 CRYPTO_DRBG_HASH_SHA512_Init, **1717**
 CRYPTO_DRBG_HASH_SHA512_Reseed, **1718**
 CRYPTO_DRBG_HMAC_SHA1_CAVS_SelfTest, **1739**
 CRYPTO_DRBG_HMAC_SHA1_Get, **1737**
 CRYPTO_DRBG_HMAC_SHA1_Init, **1735**
 CRYPTO_DRBG_HMAC_SHA1_Reseed, **1736**
 CRYPTO_DRBG_HMAC_SHA224_CAVS_SelfTest, **1745**
 CRYPTO_DRBG_HMAC_SHA224_Get, **1743**
 CRYPTO_DRBG_HMAC_SHA224_Init, **1741**
 CRYPTO_DRBG_HMAC_SHA224_Reseed, **1742**
 CRYPTO_DRBG_HMAC_SHA256_CAVS_SelfTest, **1751**
 CRYPTO_DRBG_HMAC_SHA256_Get, **1749**
 CRYPTO_DRBG_HMAC_SHA256_Init, **1747**
 CRYPTO_DRBG_HMAC_SHA256_Reseed, **1748**
 CRYPTO_DRBG_HMAC_SHA384_CAVS_SelfTest, **1757**
 CRYPTO_DRBG_HMAC_SHA384_Get, **1755**
 CRYPTO_DRBG_HMAC_SHA384_Init, **1753**
 CRYPTO_DRBG_HMAC_SHA384_Reseed, **1754**
 CRYPTO_DRBG_HMAC_SHA512_224_CAVS_SelfTest, **1769**
 CRYPTO_DRBG_HMAC_SHA512_224_Get, **1767**
 CRYPTO_DRBG_HMAC_SHA512_224_Init, **1765**
 CRYPTO_DRBG_HMAC_SHA512_224_Reseed, **1766**
 CRYPTO_DRBG_HMAC_SHA512_256_CAVS_SelfTest, **1775**
 CRYPTO_DRBG_HMAC_SHA512_256_Get, **1773**
 CRYPTO_DRBG_HMAC_SHA512_256_Init, **1771**

CRYPTO_DRBG_HMAC_SHA512_256_Reseed, **1772**
 CRYPTO_DRBG_HMAC_SHA512_CAVS_SelfTest, **1763**
 CRYPTO_DRBG_HMAC_SHA512_Get, **1761**
 CRYPTO_DRBG_HMAC_SHA512_Init, **1759**
 CRYPTO_DRBG_HMAC_SHA512_Reseed, **1760**
 CRYPTO_DSA_FIPS186_GenDomainParas, **2119**
 CRYPTO_DSA_FIPS186_ValidateParas, **2120**
 CRYPTO_DSA_GenDomainParas, **2121**
 CRYPTO_DSA_GenKeys, **2122**
 CRYPTO_DSA_InitDomainParams, **2110**
 CRYPTO_DSA_InitPrivateKey, **2111**
 CRYPTO_DSA_InitPublicKey, **2112**
 CRYPTO_DSA_InitSignature, **2113**
 CRYPTO_DSA_IsValidSignature, **2125**
 CRYPTO_DSA_KillDomainParams, **2114**
 CRYPTO_DSA_KillPrivateKey, **2115**
 CRYPTO_DSA_KillPublicKey, **2116**
 CRYPTO_DSA_KillSignature, **2117**
 CRYPTO_DSA RFC6979_SHA1_Sign, **2126**
 CRYPTO_DSA RFC6979_SHA224_Sign, **2127**
 CRYPTO_DSA RFC6979_SHA256_Sign, **2128**
 CRYPTO_DSA RFC6979_SHA384_Sign, **2129**
 CRYPTO_DSA RFC6979_SHA3_224_Sign, **2133**
 CRYPTO_DSA RFC6979_SHA3_256_Sign, **2134**
 CRYPTO_DSA RFC6979_SHA3_384_Sign, **2135**
 CRYPTO_DSA RFC6979_SHA3_512_Sign, **2136**
 CRYPTO_DSA RFC6979_SHA512_224_Sign, **2131**
 CRYPTO_DSA RFC6979_SHA512_256_Sign, **2132**
 CRYPTO_DSA RFC6979_SHA512_Sign, **2130**
 CRYPTO_DSA SEGGER_SelfTest, **2163**
 CRYPTO_DSA SHA1_CAVS_SelfTest, **2164**
 CRYPTO_DSA SHA1_Sign, **2137**
 CRYPTO_DSA SHA1_Verify, **2151**
 CRYPTO_DSA SHA224_CAVS_SelfTest, **2165**
 CRYPTO_DSA SHA224_Sign, **2138**
 CRYPTO_DSA SHA224_Verify, **2152**
 CRYPTO_DSA SHA256_CAVS_SelfTest, **2166**
 CRYPTO_DSA SHA256_Sign, **2139**
 CRYPTO_DSA SHA256_Verify, **2153**
 CRYPTO_DSA SHA384_CAVS_SelfTest, **2167**
 CRYPTO_DSA SHA384_Sign, **2140**
 CRYPTO_DSA SHA384_Verify, **2154**
 CRYPTO_DSA SHA3_224_Sign, **2144**
 CRYPTO_DSA SHA3_224_Verify, **2158**
 CRYPTO_DSA SHA3_256_Sign, **2145**
 CRYPTO_DSA SHA3_256_Verify, **2159**
 CRYPTO_DSA SHA3_384_Sign, **2146**
 CRYPTO_DSA SHA3_384_Verify, **2160**
 CRYPTO_DSA SHA3_512_Sign, **2147**
 CRYPTO_DSA SHA3_512_Verify, **2161**
 CRYPTO_DSA SHA512_224_Sign, **2142**
 CRYPTO_DSA SHA512_224_Verify, **2156**
 CRYPTO_DSA SHA512_256_Sign, **2143**
 CRYPTO_DSA SHA512_256_Verify, **2157**
 CRYPTO_DSA SHA512_CAVS_SelfTest, **2168**
 CRYPTO_DSA SHA512_Sign, **2141**
 CRYPTO_DSA SHA512_Verify, **2155**
 CRYPTO_DSA_SignDigest, **2148**
 CRYPTO_DSA_SignDigestWithK, **2149**
 CRYPTO_DSA_VerifyDigest, **2150**
 CRYPTO_EC_Add_Affine, **2404**
 CRYPTO_EC_Add_Projective, **2405**
 CRYPTO_EC_AssignInf, **2406**
 CRYPTO_EC_AssignPoint, **2407**, 2799
 CRYPTO_EC_EvictPoint, **2408**
 CRYPTO_EC_InitCurve, **2409**
 CRYPTO_EC_InitPoint, **2410**, 2799
 CRYPTO_EC_KillCurve, **2411**
 CRYPTO_EC_KillPoint, **2412**, 2799
 CRYPTO_EC_MakeAffine, **2413**
 CRYPTO_EC_MakeProjective, **2414**, 2799
 CRYPTO_EC_MovePoint, **2415**
 CRYPTO_EC_Mul, **2416**, 2797
 CRYPTO_EC_Mul2_Affine, **2435**
 CRYPTO_EC_Mul2_Projective, **2436**
 CRYPTO_EC_Mul2Pow_Projective, **2437**

CRYPTO_EC_Mul_2b_FW, **2417**, 2797
 CRYPTO_EC_Mul_2b_RM, **2422**, 2797
 CRYPTO_EC_Mul_2w_NAF, **2427**, 2797
 CRYPTO_EC_Mul_3b_FW, **2418**, 2797
 CRYPTO_EC_Mul_3b_RM, **2423**, 2797
 CRYPTO_EC_Mul_3w_NAF, **2428**, 2797
 CRYPTO_EC_Mul_4b_FW, **2419**, 2797
 CRYPTO_EC_Mul_4b_RM, **2424**, 2797
 CRYPTO_EC_Mul_4w_NAF, **2429**, 2797
 CRYPTO_EC_Mul_5b_FW, **2420**, 2797
 CRYPTO_EC_Mul_5b_RM, **2425**, 2797
 CRYPTO_EC_Mul_5w_NAF, **2430**, 2797
 CRYPTO_EC_Mul_6b_FW, **2421**, 2797
 CRYPTO_EC_Mul_6b_RM, **2426**, 2797
 CRYPTO_EC_Mul_6w_NAF, **2431**, 2797
 CRYPTO_EC_Mul_Affine, **2433**
 CRYPTO_EC_Mul_Basic, **2432**, 2797
 CRYPTO_EC_Mul_Projective, **2434**
 CRYPTO_EC_NSA_SelfTest, **2444**
 CRYPTO_EC_RFC7027_SelfTest, **2445**
 CRYPTO_EC SEGGER_SelfTest, **2446**
 CRYPTO_EC_Sub_Projective, **2438**
 CRYPTO_EC_TwinMul, **2439**
 CRYPTO_EC_WrPointCompressed, **2441**
 CRYPTO_EC_WrPointHybrid, **2442**
 CRYPTO_EC_WrPointUncompressed, **2440**
 CRYPTO_ECC_ModDiv, **2401**
 CRYPTO_ECC_ModMul, **2402**
 CRYPTO_ECC_ModSquare, **2403**
 CRYPTO_ECDH_KA_Agree, **2384**, 2390, 2390
 CRYPTO_ECDH_KA_Init, **2381**, 2390, 2390
 CRYPTO_ECDH_KA_Kill, **2385**, 2390, 2390, 2391, 2391
 CRYPTO_ECDH_KA SEGGER_SelfTest, **2387**
 CRYPTO_ECDH_KA_Start, **2382**, 2390, 2390
 CRYPTO_ECDH_KA_StartEx, **2383**
 CRYPTO_ECDSA_GenKeys, **2179**, 2248, 2249
 CRYPTO_ECDSA_InitPrivateKey, 2169, **2172**, 2248, 2249
 CRYPTO_ECDSA_InitPublicKey, 2170, **2173**, 2248, 2249
 CRYPTO_ECDSA_InitSignature, 2169, 2170, **2174**, 2248, 2249
 CRYPTO_ECDSA_KillPrivateKey, **2175**, 2248, 2249
 CRYPTO_ECDSA_KillPublicKey, **2176**, 2248, 2249
 CRYPTO_ECDSA_KillSignature, **2177**, 2248, 2248, 2249
 CRYPTO_ECDSA_PKV_CAVS_SelfTest, **2243**
 CRYPTO_ECDSA_RFC6979_SelfTest, **2246**
 CRYPTO_ECDSA_RFC6979_SHA1_GenK, **2184**
 CRYPTO_ECDSA_RFC6979_SHA1_Sign, **2195**
 CRYPTO_ECDSA_RFC6979_SHA1_SignDigest, **2206**
 CRYPTO_ECDSA_RFC6979_SHA224_GenK, **2185**
 CRYPTO_ECDSA_RFC6979_SHA224_Sign, **2196**
 CRYPTO_ECDSA_RFC6979_SHA224_SignDigest, **2207**
 CRYPTO_ECDSA_RFC6979_SHA256_GenK, **2186**
 CRYPTO_ECDSA_RFC6979_SHA256_Sign, **2197**
 CRYPTO_ECDSA_RFC6979_SHA256_SignDigest, **2208**
 CRYPTO_ECDSA_RFC6979_SHA384_GenK, **2187**
 CRYPTO_ECDSA_RFC6979_SHA384_Sign, **2198**
 CRYPTO_ECDSA_RFC6979_SHA384_SignDigest, **2209**
 CRYPTO_ECDSA_RFC6979_SHA3_224_GenK, **2191**
 CRYPTO_ECDSA_RFC6979_SHA3_224_Sign, **2202**
 CRYPTO_ECDSA_RFC6979_SHA3_224_SignDigest, **2213**
 CRYPTO_ECDSA_RFC6979_SHA3_256_GenK, **2192**
 CRYPTO_ECDSA_RFC6979_SHA3_256_Sign, **2203**
 CRYPTO_ECDSA_RFC6979_SHA3_256_SignDigest, **2214**
 CRYPTO_ECDSA_RFC6979_SHA3_384_GenK, **2193**
 CRYPTO_ECDSA_RFC6979_SHA3_384_Sign, **2204**
 CRYPTO_ECDSA_RFC6979_SHA3_384_SignDigest, **2215**
 CRYPTO_ECDSA_RFC6979_SHA3_512_GenK, **2194**
 CRYPTO_ECDSA_RFC6979_SHA3_512_Sign, **2205**
 CRYPTO_ECDSA_RFC6979_SHA3_512_SignDigest, **2216**
 CRYPTO_ECDSA_RFC6979_SHA512_224_GenK, **2189**
 CRYPTO_ECDSA_RFC6979_SHA512_224_Sign, **2200**
 CRYPTO_ECDSA_RFC6979_SHA512_224_SignDigest, **2211**
 CRYPTO_ECDSA_RFC6979_SHA512_256_GenK, **2190**
 CRYPTO_ECDSA_RFC6979_SHA512_256_Sign, **2201**
 CRYPTO_ECDSA_RFC6979_SHA512_256_SignDigest, **2212**
 CRYPTO_ECDSA_RFC6979_SHA512_GenK, **2188**
 CRYPTO_ECDSA_RFC6979_SHA512_Sign, **2199**

CRYPTO_ECDSA_RFC6979_SHA512_SignDigest, **2210**
 CRYPTO_ECDSA_SHA1_Sign, **2169**, **2217**
 CRYPTO_ECDSA_SHA1_Verify, **2170**, **2218**
 CRYPTO_ECDSA_SHA224_Sign, **2219**
 CRYPTO_ECDSA_SHA224_Verify, **2220**
 CRYPTO_ECDSA_SHA256_Sign, **2221**
 CRYPTO_ECDSA_SHA256_Verify, **2222**
 CRYPTO_ECDSA_SHA384_Sign, **2223**
 CRYPTO_ECDSA_SHA384_Verify, **2224**
 CRYPTO_ECDSA_SHA3_224_Sign, **2231**
 CRYPTO_ECDSA_SHA3_224_Verify, **2232**
 CRYPTO_ECDSA_SHA3_256_Sign, **2233**
 CRYPTO_ECDSA_SHA3_256_Verify, **2234**
 CRYPTO_ECDSA_SHA3_384_Sign, **2235**
 CRYPTO_ECDSA_SHA3_384_Verify, **2236**
 CRYPTO_ECDSA_SHA3_512_Sign, **2237**
 CRYPTO_ECDSA_SHA3_512_Verify, **2238**
 CRYPTO_ECDSA_SHA512_224_Sign, **2225**
 CRYPTO_ECDSA_SHA512_224_Verify, **2226**
 CRYPTO_ECDSA_SHA512_256_Sign, **2227**
 CRYPTO_ECDSA_SHA512_256_Verify, **2228**
 CRYPTO_ECDSA_SHA512_Sign, **2229**
 CRYPTO_ECDSA_SHA512_Verify, **2230**
 CRYPTO_ECDSA_Sign_CAVS_SelfTest, **2244**
 CRYPTO_ECDSA_SignDigest, **2239**, 2248, 2249
 CRYPTO_ECDSA_SignDigestWithK, **2240**
 CRYPTO_ECDSA_Verify_CAVS_SelfTest, **2245**
 CRYPTO_ECDSA_VerifyDigest, **2241**, 2249
 CRYPTO_EdDSA_Ed25519_Bernstein_SelfTest, **2294**
 CRYPTO_EdDSA_Ed25519_CalcPublicKey, **2266**
 CRYPTO_EdDSA_Ed25519_GenKeys, **2267**, 2298, 2300
 CRYPTO_EdDSA_Ed25519_GenKeysEx, **2268**
 CRYPTO_EdDSA_Ed25519_RdPrivateKey, **2273**
 CRYPTO_EdDSA_Ed25519_RdPublicKey, **2271**
 CRYPTO_EdDSA_Ed25519_RdSignature, **2269**
 CRYPTO_EdDSA_Ed25519_RFC8032_SelfTest, **2295**
 CRYPTO_EdDSA_Ed25519_Sign, **2260**, 2299, 2300
 CRYPTO_EdDSA_Ed25519_SignDigest, **2262**
 CRYPTO_EdDSA_Ed25519_SignEx, **2261**
 CRYPTO_EdDSA_Ed25519_Verify, **2263**, 2300
 CRYPTO_EdDSA_Ed25519_VerifyDigest, **2265**
 CRYPTO_EdDSA_Ed25519_VerifyEx, **2264**
 CRYPTO_EdDSA_Ed25519_WrPrivateKey, **2274**
 CRYPTO_EdDSA_Ed25519_WrPublicKey, **2272**
 CRYPTO_EdDSA_Ed25519_WrSignature, **2270**
 CRYPTO_EdDSA_Ed448_CalcPublicKey, **2286**, 2301
 CRYPTO_EdDSA_Ed448_GenKeys, **2284**
 CRYPTO_EdDSA_Ed448_GenKeysEx, **2285**
 CRYPTO_EdDSA_Ed448_RdPrivateKey, **2289**
 CRYPTO_EdDSA_Ed448_RdPublicKey, **2287**
 CRYPTO_EdDSA_Ed448_RdSignature, **2291**
 CRYPTO_EdDSA_Ed448_RFC8032_SelfTest, **2296**
 CRYPTO_EdDSA_Ed448_Sign, **2276**, 2299, 2301
 CRYPTO_EdDSA_Ed448_SignDigest, **2278**
 CRYPTO_EdDSA_Ed448_SignDigestEx, **2279**
 CRYPTO_EdDSA_Ed448_SignEx, **2277**
 CRYPTO_EdDSA_Ed448_Verify, **2280**, 2301
 CRYPTO_EdDSA_Ed448_VerifyDigest, **2282**
 CRYPTO_EdDSA_Ed448_VerifyDigestEx, **2283**
 CRYPTO_EdDSA_Ed448_VerifyEx, **2281**
 CRYPTO_EdDSA_Ed448_WrPrivateKey, **2290**
 CRYPTO_EdDSA_Ed448_WrPublicKey, **2288**
 CRYPTO_EdDSA_Ed448_WrSignature, **2292**
 CRYPTO_EdDSA_InitPrivateKey, **2253**, 2298, 2299, 2300, 2301
 CRYPTO_EdDSA_InitPublicKey, **2254**, 2298, 2300, 2301
 CRYPTO_EdDSA_InitSignature, **2255**, 2298, 2299, 2300, 2301
 CRYPTO_EdDSA_KillPrivateKey, **2256**, 2299, 2299, 2300, 2301
 CRYPTO_EdDSA_KillPublicKey, **2257**, 2299, 2300, 2301
 CRYPTO_EdDSA_KillSignature, **2258**, 2299, 2299, 2299, 2299, 2300, 2301
 CRYPTO_FORTUNA_Add, **1683**, 2803
 CRYPTO_FORTUNA_AddEx, **1684**
 CRYPTO_FORTUNA_Get, **1685**, 2803
 CRYPTO_FORTUNA_Init, **1686**, 2803
 CRYPTO_FORTUNA_Kill, **1687**
 CRYPTO_FORTUNA_Reseed, **1688**
 CRYPTO_FORTUNA_Status, **1689**, 2803

CRYPTO_FORTUNA_Voss_SelfTest, **1691**
 CRYPTO_GetCopyrightText, **44**, 115, 140, 167, 214, 258, 2250, 2302, 2391, 2758, 2763, 2768, 2770, 2773, 2776, 2794, 2801, 2805
 CRYPTO_GetVersionText, **45**, 115, 140, 167, 214, 258, 2250, 2302, 2391, 2758, 2763, 2768, 2771, 2774, 2777, 2794, 2801, 2805
 CRYPTO_GHASH_Add, **404**
 CRYPTO_GHASH_Calc, **405**
 CRYPTO_GHASH_Final, **406**
 CRYPTO_GHASH_InitEx, **407**
 CRYPTO_GHASH_Kill, **408**
 CRYPTO_GMAC_AES_Add, **586**
 CRYPTO_GMAC_AES_Calc, **587**
 CRYPTO_GMAC_AES_Calc_128, **588**
 CRYPTO_GMAC_AES_Final, **589**
 CRYPTO_GMAC_AES_Final_128, **590**
 CRYPTO_GMAC_AES_InitEx, **591**
 CRYPTO_GMAC_AES_Kill, **592**
 CRYPTO_GMAC_ARIA_Add, **616**
 CRYPTO_GMAC_ARIA_Calc, **617**
 CRYPTO_GMAC_ARIA_Calc_128, **618**
 CRYPTO_GMAC_ARIA_Final, **619**
 CRYPTO_GMAC_ARIA_Final_128, **620**
 CRYPTO_GMAC_ARIA_InitEx, **621**
 CRYPTO_GMAC_ARIA_Kill, **622**
 CRYPTO_GMAC_CAMELLIA_Add, **631**
 CRYPTO_GMAC_CAMELLIA_Calc, **632**
 CRYPTO_GMAC_CAMELLIA_Calc_128, **633**
 CRYPTO_GMAC_CAMELLIA_Final, **634**
 CRYPTO_GMAC_CAMELLIA_Final_128, **635**
 CRYPTO_GMAC_CAMELLIA_InitEx, **636**
 CRYPTO_GMAC_CAMELLIA_Kill, **637**
 CRYPTO_GMAC_SEED_Add, **601**
 CRYPTO_GMAC_SEED_Calc, **602**
 CRYPTO_GMAC_SEED_Calc_128, **603**
 CRYPTO_GMAC_SEED_Final, **604**
 CRYPTO_GMAC_SEED_Final_128, **605**
 CRYPTO_GMAC_SEED_InitEx, **606**
 CRYPTO_GMAC_SEED_Kill, **607**
 CRYPTO_GMAC_TWOFISH_Add, **646**
 CRYPTO_GMAC_TWOFISH_Calc, **647**
 CRYPTO_GMAC_TWOFISH_Calc_128, **648**
 CRYPTO_GMAC_TWOFISH_Final, **649**
 CRYPTO_GMAC_TWOFISH_Final_128, **650**
 CRYPTO_GMAC_TWOFISH_InitEx, **651**
 CRYPTO_GMAC_TWOFISH_Kill, **652**
 CRYPTO_HASH_BLAKE2B_Add, **63**
 CRYPTO_HASH_BLAKE2B_Final, **64**
 CRYPTO_HASH_BLAKE2B_Get, **65**
 CRYPTO_HASH_BLAKE2B_Init, **66**
 CRYPTO_HASH_BLAKE2B_Kill, **67**
 CRYPTO_HASH_BLAKE2S_Add, **85**
 CRYPTO_HASH_BLAKE2S_Final, **86**
 CRYPTO_HASH_BLAKE2S_Get, **87**
 CRYPTO_HASH_BLAKE2S_Init, **88**
 CRYPTO_HASH_BLAKE2S_Kill, **89**
 CRYPTO_HASH_MD5_Add, **107**
 CRYPTO_HASH_MD5_Final, **108**
 CRYPTO_HASH_MD5_Get, **109**
 CRYPTO_HASH_MD5_Init, **110**
 CRYPTO_HASH_MD5_Kill, **111**
 CRYPTO_HASH_RIPEMD160_Add, **132**
 CRYPTO_HASH_RIPEMD160_Final, **133**
 CRYPTO_HASH_RIPEMD160_Get, **134**
 CRYPTO_HASH_RIPEMD160_Init, **135**
 CRYPTO_HASH_RIPEMD160_Kill, **136**
 CRYPTO_HASH_SHA1_Add, **158**
 CRYPTO_HASH_SHA1_Final, **159**
 CRYPTO_HASH_SHA1_Get, **160**
 CRYPTO_HASH_SHA1_Init, **161**
 CRYPTO_HASH_SHA1_Kill, **162**
 CRYPTO_HASH_SHA224_Add, **183**
 CRYPTO_HASH_SHA224_Final, **184**
 CRYPTO_HASH_SHA224_Get, **185**
 CRYPTO_HASH_SHA224_Init, **186**
 CRYPTO_HASH_SHA224_Kill, **187**
 CRYPTO_HASH_SHA256_Add, **205**

CRYPTO_HASH_SHA256_Final, **206**
CRYPTO_HASH_SHA256_Get, **207**
CRYPTO_HASH_SHA256_Init, **208**
CRYPTO_HASH_SHA256_Kill, **209**
CRYPTO_HASH_SHA384_Add, **227**
CRYPTO_HASH_SHA384_Final, **228**
CRYPTO_HASH_SHA384_Get, **229**
CRYPTO_HASH_SHA384_Init, **230**
CRYPTO_HASH_SHA384_Kill, **231**
CRYPTO_HASH_SHA3_224_Add, **306**
CRYPTO_HASH_SHA3_224_Final, **307**
CRYPTO_HASH_SHA3_224_Get, **308**
CRYPTO_HASH_SHA3_224_Init, **309**
CRYPTO_HASH_SHA3_224_Kill, **310**
CRYPTO_HASH_SHA3_256_Add, **328**
CRYPTO_HASH_SHA3_256_Final, **329**
CRYPTO_HASH_SHA3_256_Get, **330**
CRYPTO_HASH_SHA3_256_Init, **331**
CRYPTO_HASH_SHA3_256_Kill, **332**
CRYPTO_HASH_SHA3_384_Add, **350**
CRYPTO_HASH_SHA3_384_Final, **351**
CRYPTO_HASH_SHA3_384_Get, **352**
CRYPTO_HASH_SHA3_384_Init, **353**
CRYPTO_HASH_SHA3_384_Kill, **354**
CRYPTO_HASH_SHA3_512_Add, **372**
CRYPTO_HASH_SHA3_512_Final, **373**
CRYPTO_HASH_SHA3_512_Get, **374**
CRYPTO_HASH_SHA3_512_Init, **375**
CRYPTO_HASH_SHA3_512_Kill, **376**
CRYPTO_HASH_SHA512_224_Add, **271**
CRYPTO_HASH_SHA512_224_Final, **272**
CRYPTO_HASH_SHA512_224_Get, **273**
CRYPTO_HASH_SHA512_224_Init, **274**
CRYPTO_HASH_SHA512_224_Kill, **275**
CRYPTO_HASH_SHA512_256_Add, **287**
CRYPTO_HASH_SHA512_256_Final, **288**
CRYPTO_HASH_SHA512_256_Get, **289**
CRYPTO_HASH_SHA512_256_Init, **290**
CRYPTO_HASH_SHA512_256_Kill, **291**
CRYPTO_HASH_SHA512_Add, **249**
CRYPTO_HASH_SHA512_Final, **250**
CRYPTO_HASH_SHA512_Get, **251**
CRYPTO_HASH_SHA512_Init, **252**
CRYPTO_HASH_SHA512_Kill, **253**
CRYPTO_HASH_SM3_Add, **395**
CRYPTO_HASH_SM3_Final, **396**
CRYPTO_HASH_SM3_Get, **397**
CRYPTO_HASH_SM3_Init, **398**
CRYPTO_HASH_SM3_Kill, **399**
CRYPTO_HKDF_MD5_Calc, **1884**
CRYPTO_HKDF_MD5_Expand, **1885**
CRYPTO_HKDF_MD5_Extract, **1886**
CRYPTO_HKDF_RIPEMD160_Calc, **1887**
CRYPTO_HKDF_RIPEMD160_Expand, **1888**
CRYPTO_HKDF_RIPEMD160_Extract, **1889**
CRYPTO_HKDF_SelfTest, **1915**
CRYPTO_HKDF_SHA1_Calc, **1890**
CRYPTO_HKDF_SHA1_Expand, **1891**
CRYPTO_HKDF_SHA1_Extract, **1892**
CRYPTO_HKDF_SHA224_Calc, **1893**
CRYPTO_HKDF_SHA224_Expand, **1894**
CRYPTO_HKDF_SHA224_Extract, **1895**
CRYPTO_HKDF_SHA256_Calc, **1896**
CRYPTO_HKDF_SHA256_Expand, **1897**
CRYPTO_HKDF_SHA256_Extract, **1898**
CRYPTO_HKDF_SHA384_Calc, **1899**
CRYPTO_HKDF_SHA384_Expand, **1900**
CRYPTO_HKDF_SHA384_Extract, **1901**
CRYPTO_HKDF_SHA512_224_Calc, **1902**
CRYPTO_HKDF_SHA512_224_Expand, **1903**
CRYPTO_HKDF_SHA512_224_Extract, **1904**
CRYPTO_HKDF_SHA512_256_Calc, **1905**
CRYPTO_HKDF_SHA512_256_Expand, **1906**
CRYPTO_HKDF_SHA512_256_Extract, **1907**
CRYPTO_HKDF_SHA512_Calc, **1908**
CRYPTO_HKDF_SHA512_Expand, **1909**

CRYPTO_HKDF_SHA512_Extract, **1910**
 CRYPTO_HKDF_SM3_Calc, **1911**
 CRYPTO_HKDF_SM3_Expand, **1912**
 CRYPTO_HKDF_SM3_Extract, **1913**
 CRYPTO_HMAC_MD5_Add, **661**
 CRYPTO_HMAC_MD5_Calc, **662**
 CRYPTO_HMAC_MD5_Calc_160, **663**
 CRYPTO_HMAC_MD5_Final, **664**
 CRYPTO_HMAC_MD5_Final_160, **665**
 CRYPTO_HMAC_MD5_Init, **666**
 CRYPTO_HMAC_MD5_InitEx, **667**
 CRYPTO_HMAC_MD5_Kill, **668**
 CRYPTO_HMAC_MD5_Reset, **669**
 CRYPTO_HMAC_RIPEMD160_Add, **680**
 CRYPTO_HMAC_RIPEMD160_Calc, **681**
 CRYPTO_HMAC_RIPEMD160_Calc_160, **682**
 CRYPTO_HMAC_RIPEMD160_Final, **683**
 CRYPTO_HMAC_RIPEMD160_Final_160, **684**
 CRYPTO_HMAC_RIPEMD160_Init, **685**
 CRYPTO_HMAC_RIPEMD160_InitEx, **686**
 CRYPTO_HMAC_RIPEMD160_Kill, **687**
 CRYPTO_HMAC_RIPEMD160_Reset, **688**
 CRYPTO_HMAC_SHA1_Add, 411, 411, **698**
 CRYPTO_HMAC_SHA1_Calc, **699**
 CRYPTO_HMAC_SHA1_Calc_160, **700**
 CRYPTO_HMAC_SHA1_CAVS_SelfTest, **717**
 CRYPTO_HMAC_SHA1_Final, 411, **701**
 CRYPTO_HMAC_SHA1_Final_160, **702**
 CRYPTO_HMAC_SHA1_Init, 411, **703**
 CRYPTO_HMAC_SHA1_InitEx, **704**
 CRYPTO_HMAC_SHA1_Kill, **705**
 CRYPTO_HMAC_SHA1_Reset, **706**
 CRYPTO_HMAC_SHA1_RFC2202_SelfTest, **716**
 CRYPTO_HMAC_SHA224_Add, **720**
 CRYPTO_HMAC_SHA224_Calc, **721**
 CRYPTO_HMAC_SHA224_Calc_224, **722**
 CRYPTO_HMAC_SHA224_CAVS_SelfTest, **737**
 CRYPTO_HMAC_SHA224_Final, **723**
 CRYPTO_HMAC_SHA224_Final_224, **724**
 CRYPTO_HMAC_SHA224_Init, **725**
 CRYPTO_HMAC_SHA224_InitEx, **726**
 CRYPTO_HMAC_SHA224_Kill, **727**
 CRYPTO_HMAC_SHA224_Reset, **728**
 CRYPTO_HMAC_SHA256_Add, **740**
 CRYPTO_HMAC_SHA256_Calc, **741**
 CRYPTO_HMAC_SHA256_Calc_256, **742**
 CRYPTO_HMAC_SHA256_CAVS_SelfTest, **757**
 CRYPTO_HMAC_SHA256_Final, **743**
 CRYPTO_HMAC_SHA256_Final_256, **744**
 CRYPTO_HMAC_SHA256_Init, **745**
 CRYPTO_HMAC_SHA256_InitEx, **746**
 CRYPTO_HMAC_SHA256_Kill, **748**
 CRYPTO_HMAC_SHA256_Reset, **747**
 CRYPTO_HMAC_SHA256_RFC4231_SelfTest, **758**
 CRYPTO_HMAC_SHA384_Add, **761**
 CRYPTO_HMAC_SHA384_Calc, **762**
 CRYPTO_HMAC_SHA384_Calc_384, **763**
 CRYPTO_HMAC_SHA384_CAVS_SelfTest, **778**
 CRYPTO_HMAC_SHA384_Final, **764**
 CRYPTO_HMAC_SHA384_Final_384, **765**
 CRYPTO_HMAC_SHA384_Init, **766**
 CRYPTO_HMAC_SHA384_InitEx, **767**
 CRYPTO_HMAC_SHA384_Kill, **768**
 CRYPTO_HMAC_SHA384_Reset, **769**
 CRYPTO_HMAC_SHA3_224_Add, **837**
 CRYPTO_HMAC_SHA3_224_Calc, **838**
 CRYPTO_HMAC_SHA3_224_Calc_224, **839**
 CRYPTO_HMAC_SHA3_224_Final, **840**
 CRYPTO_HMAC_SHA3_224_Final_224, **841**
 CRYPTO_HMAC_SHA3_224_Init, **842**
 CRYPTO_HMAC_SHA3_224_InitEx, **843**
 CRYPTO_HMAC_SHA3_224_Kill, **844**
 CRYPTO_HMAC_SHA3_224_Reset, **845**
 CRYPTO_HMAC_SHA3_256_Add, **854**
 CRYPTO_HMAC_SHA3_256_Calc, **855**
 CRYPTO_HMAC_SHA3_256_Calc_256, **856**

CRYPTO_HMAC_SHA3_256_Final, **857**
 CRYPTO_HMAC_SHA3_256_Final_256, **858**
 CRYPTO_HMAC_SHA3_256_Init, **859**
 CRYPTO_HMAC_SHA3_256_InitEx, **860**
 CRYPTO_HMAC_SHA3_256_Kill, **861**
 CRYPTO_HMAC_SHA3_256_Reset, **862**
 CRYPTO_HMAC_SHA3_384_Add, **871**
 CRYPTO_HMAC_SHA3_384_Calc, **872**
 CRYPTO_HMAC_SHA3_384_Calc_384, **873**
 CRYPTO_HMAC_SHA3_384_Final, **874**
 CRYPTO_HMAC_SHA3_384_Final_384, **875**
 CRYPTO_HMAC_SHA3_384_Init, **876**
 CRYPTO_HMAC_SHA3_384_InitEx, **877**
 CRYPTO_HMAC_SHA3_384_Kill, **878**
 CRYPTO_HMAC_SHA3_384_Reset, **879**
 CRYPTO_HMAC_SHA3_512_Add, **888**
 CRYPTO_HMAC_SHA3_512_Calc, **889**
 CRYPTO_HMAC_SHA3_512_Calc_512, **890**
 CRYPTO_HMAC_SHA3_512_Final, **891**
 CRYPTO_HMAC_SHA3_512_Final_512, **892**
 CRYPTO_HMAC_SHA3_512_Init, **893**
 CRYPTO_HMAC_SHA3_512_Kill, **894**
 CRYPTO_HMAC_SHA3_512_Reset, **895**
 CRYPTO_HMAC_SHA512_224_Add, **801**
 CRYPTO_HMAC_SHA512_224_Calc, **802**
 CRYPTO_HMAC_SHA512_224_Calc_224, **803**
 CRYPTO_HMAC_SHA512_224_Final, **804**
 CRYPTO_HMAC_SHA512_224_Final_224, **805**
 CRYPTO_HMAC_SHA512_224_Init, **806**
 CRYPTO_HMAC_SHA512_224_InitEx, **807**
 CRYPTO_HMAC_SHA512_224_Kill, **808**
 CRYPTO_HMAC_SHA512_224_Reset, **809**
 CRYPTO_HMAC_SHA512_256_Add, **819**
 CRYPTO_HMAC_SHA512_256_Calc, **820**
 CRYPTO_HMAC_SHA512_256_Calc_256, **821**
 CRYPTO_HMAC_SHA512_256_Final, **822**
 CRYPTO_HMAC_SHA512_256_Final_256, **823**
 CRYPTO_HMAC_SHA512_256_Init, **824**
 CRYPTO_HMAC_SHA512_256_InitEx, **825**
 CRYPTO_HMAC_SHA512_256_Kill, **826**
 CRYPTO_HMAC_SHA512_256_Reset, **827**
 CRYPTO_HMAC_SHA512_Add, **781**
 CRYPTO_HMAC_SHA512_Calc, **782**
 CRYPTO_HMAC_SHA512_Calc_512, **783**
 CRYPTO_HMAC_SHA512_CAVS_SelfTest, **797**
 CRYPTO_HMAC_SHA512_Final, **784**
 CRYPTO_HMAC_SHA512_Final_512, **785**
 CRYPTO_HMAC_SHA512_Init, **786**
 CRYPTO_HMAC_SHA512_InitEx, **787**
 CRYPTO_HMAC_SHA512_Kill, **788**
 CRYPTO_HMAC_SHA512_Reset, **789**
 CRYPTO_HMAC_SHA512_RFC4231_SelfTest, **798**
 CRYPTO_HMAC_SM3_Add, **904**
 CRYPTO_HMAC_SM3_Calc, **905**
 CRYPTO_HMAC_SM3_Calc_256, **906**
 CRYPTO_HMAC_SM3_Final, **907**
 CRYPTO_HMAC_SM3_Final_256, **908**
 CRYPTO_HMAC_SM3_Init, **909**
 CRYPTO_HMAC_SM3_InitEx, **910**
 CRYPTO_HMAC_SM3_Kill, **912**
 CRYPTO_HMAC_SM3_Reset, **911**
 CRYPTO_IDEA_Ascrypt_SelfTest, **1292**
 CRYPTO_IDEA_CBC_Decrypt, **1267**
 CRYPTO_IDEA_CBC_Encrypt, **1266**
 CRYPTO_IDEA_CTR_Decrypt, **1269**
 CRYPTO_IDEA_CTR_Encrypt, **1268**
 CRYPTO_IDEA_Decrypt, **1263**
 CRYPTO_IDEA_ECB_Decrypt, **1265**
 CRYPTO_IDEA_ECB_Encrypt, **1264**
 CRYPTO_IDEA_Encrypt, **1262**
 CRYPTO_IDEA_InitDecrypt, **1260**
 CRYPTO_IDEA_InitEncrypt, **1259**
 CRYPTO_IDEA_Install, **1256**
 CRYPTO_IDEA_IsInstalled, **1257**
 CRYPTO_IDEA_Kill, **1261**
 CRYPTO_IDEA_OFB_Decrypt, **1271**

CRYPTO_IDEA_OFB_Encrypt, **1270**
 CRYPTO_IDEA_QueryInstall, **1258**
 CRYPTO_IncCTRBE, **2749**
 CRYPTO_IncCTRBE_TrapOverflow, **2750**
 CRYPTO_IncCTRLE, **2751**
 CRYPTO_IncCTRLE_TrapOverflow, **2752**
 CRYPTO_Init, 30, **46**, 115, 140, 167, 214, 258, 2250, 2302, 2391, 2758, 2763, 2768, 2770, 2773, 2776, 2794, 2801, 2804
 CRYPTO_KDF1_BLAKE2B_Calc, **1827**
 CRYPTO_KDF1_BLAKE2B_CalcEx, **1828**
 CRYPTO_KDF1_BLAKE2S_Calc, **1829**
 CRYPTO_KDF1_BLAKE2S_CalcEx, **1830**
 CRYPTO_KDF1_SHA1_Calc, **1803**
 CRYPTO_KDF1_SHA1_CalcEx, **1804**
 CRYPTO_KDF1_SHA224_Calc, **1805**
 CRYPTO_KDF1_SHA224_CalcEx, **1806**
 CRYPTO_KDF1_SHA256_Calc, **1807**
 CRYPTO_KDF1_SHA256_CalcEx, **1808**
 CRYPTO_KDF1_SHA384_Calc, **1809**
 CRYPTO_KDF1_SHA384_CalcEx, **1810**
 CRYPTO_KDF1_SHA3_224_Calc, **1817**
 CRYPTO_KDF1_SHA3_224_CalcEx, **1818**
 CRYPTO_KDF1_SHA3_256_Calc, **1819**
 CRYPTO_KDF1_SHA3_256_CalcEx, **1820**
 CRYPTO_KDF1_SHA3_384_Calc, **1821**
 CRYPTO_KDF1_SHA3_384_CalcEx, **1822**
 CRYPTO_KDF1_SHA3_512_Calc, **1823**
 CRYPTO_KDF1_SHA3_512_CalcEx, **1824**
 CRYPTO_KDF1_SHA512_224_Calc, **1813**
 CRYPTO_KDF1_SHA512_224_CalcEx, **1814**
 CRYPTO_KDF1_SHA512_256_Calc, **1815**
 CRYPTO_KDF1_SHA512_256_CalcEx, **1816**
 CRYPTO_KDF1_SHA512_Calc, **1811**
 CRYPTO_KDF1_SHA512_CalcEx, **1812**
 CRYPTO_KDF1_SM3_Calc, **1825**
 CRYPTO_KDF1_SM3_CalcEx, **1826**
 CRYPTO_KDF2_BLAKE2B_Calc, **1857**
 CRYPTO_KDF2_BLAKE2B_CalcEx, **1858**
 CRYPTO_KDF2_BLAKE2S_Calc, **1859**
 CRYPTO_KDF2_BLAKE2S_CalcEx, **1860**
 CRYPTO_KDF2_SHA1_Calc, **1833**
 CRYPTO_KDF2_SHA1_CalcEx, **1834**
 CRYPTO_KDF2_SHA224_Calc, **1835**
 CRYPTO_KDF2_SHA224_CalcEx, **1836**
 CRYPTO_KDF2_SHA256_Calc, **1837**
 CRYPTO_KDF2_SHA256_CalcEx, **1838**
 CRYPTO_KDF2_SHA384_Calc, **1839**
 CRYPTO_KDF2_SHA384_CalcEx, **1840**
 CRYPTO_KDF2_SHA3_224_Calc, **1847**
 CRYPTO_KDF2_SHA3_224_CalcEx, **1848**
 CRYPTO_KDF2_SHA3_256_Calc, **1849**
 CRYPTO_KDF2_SHA3_256_CalcEx, **1850**
 CRYPTO_KDF2_SHA3_384_Calc, **1851**
 CRYPTO_KDF2_SHA3_384_CalcEx, **1852**
 CRYPTO_KDF2_SHA3_512_Calc, **1853**
 CRYPTO_KDF2_SHA3_512_CalcEx, **1854**
 CRYPTO_KDF2_SHA512_224_Calc, **1843**
 CRYPTO_KDF2_SHA512_224_CalcEx, **1844**
 CRYPTO_KDF2_SHA512_256_Calc, **1845**
 CRYPTO_KDF2_SHA512_256_CalcEx, **1846**
 CRYPTO_KDF2_SHA512_Calc, **1841**
 CRYPTO_KDF2_SHA512_CalcEx, **1842**
 CRYPTO_KDF2_SM3_Calc, **1855**
 CRYPTO_KDF2_SM3_CalcEx, **1856**
 CRYPTO KECCAK_Add, **1957**
 CRYPTO KECCAK_AddPadding, **1958**
 CRYPTO KECCAK_Get, **1959**
 CRYPTO KECCAK_Init, **1956**
 CRYPTO KECCAK_Kill, **1960**
 CRYPTO_KMAC_128_Init, **1015**
 CRYPTO_KMAC_256_Init, **1016**
 CRYPTO_KMAC_Add, **1017**
 CRYPTO_KMAC_CSRC_SelfTest, **1020**
 CRYPTO_KMAC_Get, **1018**
 CRYPTO_KMAC_Init, **1014**
 CRYPTO_KW_AESKW_Unwrap, **2333**

CRYPTO_KW_AESKW_Wrap, **2332**
 CRYPTO_KW_AESKWP_Unwrap, **2335**
 CRYPTO_KW_AESKWP_Wrap, **2334**
 CRYPTO_KW_ARIAKW_Unwrap, **2349**
 CRYPTO_KW_ARIAKW_Wrap, **2348**
 CRYPTO_KW_ARIAKWP_Unwrap, **2351**
 CRYPTO_KW_ARIAKWP_Wrap, **2350**
 CRYPTO_KW_CAMELLIAKW_Unwrap, **2356**
 CRYPTO_KW_CAMELLIAKW_Wrap, **2355**
 CRYPTO_KW_CAMELLIAKWP_Unwrap, **2358**
 CRYPTO_KW_CAMELLIAKWP_Wrap, **2357**
 CRYPTO_KW_SEEDKW_Unwrap, **2342**
 CRYPTO_KW_SEEDKW_Wrap, **2341**
 CRYPTO_KW_SEEDKWP_Unwrap, **2344**
 CRYPTO_KW_SEEDKWP_Wrap, **2343**
 CRYPTO_KW_SP800_38F_AES_Unwrap, **2337**
 CRYPTO_KW_SP800_38F_AES_Wrap, **2336**
 CRYPTO_KW_SP800_38F_ARIA_Unwrap, **2353**
 CRYPTO_KW_SP800_38F_ARIA_Wrap, **2352**
 CRYPTO_KW_SP800_38F_CAMELLIA_Unwrap, **2360**
 CRYPTO_KW_SP800_38F_CAMELLIA_Wrap, **2359**
 CRYPTO_KW_SP800_38F_SEED_Unwrap, **2346**
 CRYPTO_KW_SP800_38F_SEED_Wrap, **2345**
 CRYPTO_KW_SP800_38F_TWOFISH_Unwrap, **2367**
 CRYPTO_KW_SP800_38F_TWOFISH_Wrap, **2366**
 CRYPTO_KW_TWOFISHKW_Unwrap, **2363**
 CRYPTO_KW_TWOFISHKW_Wrap, **2362**
 CRYPTO_KW_TWOFISHKWP_Unwrap, **2365**
 CRYPTO_KW_TWOFISHKWP_Wrap, **2364**
 CRYPTO_MAC_CMAC_AES_Add, **423**
 CRYPTO_MAC_CMAC_AES_Final, **424**
 CRYPTO_MAC_CMAC_AES_Final_128, **425**
 CRYPTO_MAC_CMAC_AES_Init, **426**
 CRYPTO_MAC_CMAC_AES_InitEx, **427**
 CRYPTO_MAC_CMAC_AES_Kill, **428**
 CRYPTO_MAC_CMAC_ARIA_Add, **511**
 CRYPTO_MAC_CMAC_ARIA_Final, **512**
 CRYPTO_MAC_CMAC_ARIA_Final_128, **513**
 CRYPTO_MAC_CMAC_ARIA_Init, **514**
 CRYPTO_MAC_CMAC_ARIA_InitEx, **515**
 CRYPTO_MAC_CMAC_ARIA_Kill, **516**
 CRYPTO_MAC_CMAC_BLOWFISH_Add, **561**
 CRYPTO_MAC_CMAC_BLOWFISH_Final, **562**
 CRYPTO_MAC_CMAC_BLOWFISH_Final_64, **563**
 CRYPTO_MAC_CMAC_BLOWFISH_Init, **564**
 CRYPTO_MAC_CMAC_BLOWFISH_InitEx, **565**
 CRYPTO_MAC_CMAC_BLOWFISH_Kill, **566**
 CRYPTO_MAC_CMAC_CAMELLIA_Add, **528**
 CRYPTO_MAC_CMAC_CAMELLIA_Final, **529**
 CRYPTO_MAC_CMAC_CAMELLIA_Final_128, **530**
 CRYPTO_MAC_CMAC_CAMELLIA_Init, **531**
 CRYPTO_MAC_CMAC_CAMELLIA_InitEx, **532**
 CRYPTO_MAC_CMAC_CAMELLIA_Kill, **533**
 CRYPTO_MAC_CMAC_CAST_Add, **477**
 CRYPTO_MAC_CMAC_CAST_Final, **478**
 CRYPTO_MAC_CMAC_CAST_Final_64, **479**
 CRYPTO_MAC_CMAC_CAST_Init, **480**
 CRYPTO_MAC_CMAC_CAST_InitEx, **481**
 CRYPTO_MAC_CMAC_CAST_Kill, **482**
 CRYPTO_MAC_CMAC_IDEA_Add, **461**
 CRYPTO_MAC_CMAC_IDEA_Final, **462**
 CRYPTO_MAC_CMAC_IDEA_Final_64, **463**
 CRYPTO_MAC_CMAC_IDEA_Init, **464**
 CRYPTO_MAC_CMAC_IDEA_InitEx, **465**
 CRYPTO_MAC_CMAC_IDEA_Kill, **466**
 CRYPTO_MAC_CMAC_PRESENT_Add, **578**
 CRYPTO_MAC_CMAC_PRESENT_Final, **579**
 CRYPTO_MAC_CMAC_PRESENT_Final_64, **580**
 CRYPTO_MAC_CMAC_PRESENT_Init, **581**
 CRYPTO_MAC_CMAC_PRESENT_InitEx, **582**
 CRYPTO_MAC_CMAC_PRESENT_Kill, **583**
 CRYPTO_MAC_CMAC_SEED_Add, **494**
 CRYPTO_MAC_CMAC_SEED_Final, **495**
 CRYPTO_MAC_CMAC_SEED_Final_128, **496**
 CRYPTO_MAC_CMAC_SEED_Init, **497**
 CRYPTO_MAC_CMAC_SEED_InitEx, **498**

CRYPTO_MAC_CMAC_SEED_Kill, **499**
 CRYPTO_MAC_CMAC_TDES_Add, **442**
 CRYPTO_MAC_CMAC_TDES_Final, **443**
 CRYPTO_MAC_CMAC_TDES_Final_64, **444**
 CRYPTO_MAC_CMAC_TDES_Init, **445**
 CRYPTO_MAC_CMAC_TDES_InitEx, **446**
 CRYPTO_MAC_CMAC_TDES_Kill, **447**
 CRYPTO_MAC_CMAC_TWOFISH_Add, **545**
 CRYPTO_MAC_CMAC_TWOFISH_Final, **546**
 CRYPTO_MAC_CMAC_TWOFISH_Final_128, **547**
 CRYPTO_MAC_CMAC_TWOFISH_Init, **548**
 CRYPTO_MAC_CMAC_TWOFISH_InitEx, **549**
 CRYPTO_MAC_CMAC_TWOFISH_Kill, **550**
 CRYPTO_MAC_GMAC_AES_Add, **594**
 CRYPTO_MAC_GMAC_AES_Final, **595**
 CRYPTO_MAC_GMAC_AES_Final_128, **596**
 CRYPTO_MAC_GMAC_AES_InitEx, **597**
 CRYPTO_MAC_GMAC_AES_Kill, **598**
 CRYPTO_MAC_GMAC_ARIA_Add, **624**
 CRYPTO_MAC_GMAC_ARIA_Final, **625**
 CRYPTO_MAC_GMAC_ARIA_Final_128, **626**
 CRYPTO_MAC_GMAC_ARIA_InitEx, **627**
 CRYPTO_MAC_GMAC_ARIA_Kill, **628**
 CRYPTO_MAC_GMAC_CAMELLIA_Add, **639**
 CRYPTO_MAC_GMAC_CAMELLIA_Final, **640**
 CRYPTO_MAC_GMAC_CAMELLIA_Final_128, **641**
 CRYPTO_MAC_GMAC_CAMELLIA_InitEx, **642**
 CRYPTO_MAC_GMAC_CAMELLIA_Kill, **643**
 CRYPTO_MAC_GMAC_SEED_Add, **609**
 CRYPTO_MAC_GMAC_SEED_Final, **610**
 CRYPTO_MAC_GMAC_SEED_Final_128, **611**
 CRYPTO_MAC_GMAC_SEED_InitEx, **612**
 CRYPTO_MAC_GMAC_SEED_Kill, **613**
 CRYPTO_MAC_GMAC_TWOFISH_Add, **654**
 CRYPTO_MAC_GMAC_TWOFISH_Final, **655**
 CRYPTO_MAC_GMAC_TWOFISH_Final_128, **656**
 CRYPTO_MAC_GMAC_TWOFISH_InitEx, **657**
 CRYPTO_MAC_GMAC_TWOFISH_Kill, **658**
 CRYPTO_MAC_HMAC_MD5_Add, **671**
 CRYPTO_MAC_HMAC_MD5_Final, **672**
 CRYPTO_MAC_HMAC_MD5_Final_160, **674**
 CRYPTO_MAC_HMAC_MD5_Final_96, **673**
 CRYPTO_MAC_HMAC_MD5_Init, **675**
 CRYPTO_MAC_HMAC_MD5_InitEx, **676**
 CRYPTO_MAC_HMAC_MD5_Kill, **677**
 CRYPTO_MAC_HMAC_RIPEMD160_Add, **690**
 CRYPTO_MAC_HMAC_RIPEMD160_Final, **691**
 CRYPTO_MAC_HMAC_RIPEMD160_Final_160, **692**
 CRYPTO_MAC_HMAC_RIPEMD160_Init, **693**
 CRYPTO_MAC_HMAC_RIPEMD160_InitEx, **694**
 CRYPTO_MAC_HMAC_RIPEMD160_Kill, **695**
 CRYPTO_MAC_HMAC_SHA1_Add, **708**
 CRYPTO_MAC_HMAC_SHA1_Final, 19, **709**
 CRYPTO_MAC_HMAC_SHA1_Final_160, 19, **711**
 CRYPTO_MAC_HMAC_SHA1_Final_96, 19, **710**
 CRYPTO_MAC_HMAC_SHA1_Init, **712**
 CRYPTO_MAC_HMAC_SHA1_InitEx, **713**
 CRYPTO_MAC_HMAC_SHA1_Kill, **714**
 CRYPTO_MAC_HMAC_SHA224_Add, **730**
 CRYPTO_MAC_HMAC_SHA224_Final, **731**
 CRYPTO_MAC_HMAC_SHA224_Final_224, **732**
 CRYPTO_MAC_HMAC_SHA224_Init, **733**
 CRYPTO_MAC_HMAC_SHA224_InitEx, **734**
 CRYPTO_MAC_HMAC_SHA224_Kill, **735**
 CRYPTO_MAC_HMAC_SHA256_Add, **750**
 CRYPTO_MAC_HMAC_SHA256_Final, **751**
 CRYPTO_MAC_HMAC_SHA256_Final_256, **752**
 CRYPTO_MAC_HMAC_SHA256_Init, **753**
 CRYPTO_MAC_HMAC_SHA256_InitEx, **754**
 CRYPTO_MAC_HMAC_SHA256_Kill, **755**
 CRYPTO_MAC_HMAC_SHA384_Add, **771**
 CRYPTO_MAC_HMAC_SHA384_Final, **772**
 CRYPTO_MAC_HMAC_SHA384_Final_384, **773**
 CRYPTO_MAC_HMAC_SHA384_Init, **774**
 CRYPTO_MAC_HMAC_SHA384_InitEx, **775**
 CRYPTO_MAC_HMAC_SHA384_Kill, **776**

CRYPTO_MAC_HMAC_SHA3_224_Add, **847**
 CRYPTO_MAC_HMAC_SHA3_224_Final, **848**
 CRYPTO_MAC_HMAC_SHA3_224_Final_224, **849**
 CRYPTO_MAC_HMAC_SHA3_224_Init, **850**
 CRYPTO_MAC_HMAC_SHA3_224_Kill, **851**
 CRYPTO_MAC_HMAC_SHA3_256_Add, **864**
 CRYPTO_MAC_HMAC_SHA3_256_Final, **865**
 CRYPTO_MAC_HMAC_SHA3_256_Final_256, **866**
 CRYPTO_MAC_HMAC_SHA3_256_Init, **867**
 CRYPTO_MAC_HMAC_SHA3_256_Kill, **868**
 CRYPTO_MAC_HMAC_SHA3_384_Add, **881**
 CRYPTO_MAC_HMAC_SHA3_384_Final, **882**
 CRYPTO_MAC_HMAC_SHA3_384_Final_384, **883**
 CRYPTO_MAC_HMAC_SHA3_384_Init, **884**
 CRYPTO_MAC_HMAC_SHA3_384_Kill, **885**
 CRYPTO_MAC_HMAC_SHA3_512_Add, **897**
 CRYPTO_MAC_HMAC_SHA3_512_Final, **898**
 CRYPTO_MAC_HMAC_SHA3_512_Final_512, **899**
 CRYPTO_MAC_HMAC_SHA3_512_Init, **900**
 CRYPTO_MAC_HMAC_SHA3_512_Kill, **901**
 CRYPTO_MAC_HMAC_SHA512_224_Add, **811**
 CRYPTO_MAC_HMAC_SHA512_224_Final, **812**
 CRYPTO_MAC_HMAC_SHA512_224_Final_224, **813**
 CRYPTO_MAC_HMAC_SHA512_224_Init, **814**
 CRYPTO_MAC_HMAC_SHA512_224_InitEx, **815**
 CRYPTO_MAC_HMAC_SHA512_224_Kill, **816**
 CRYPTO_MAC_HMAC_SHA512_256_Add, **829**
 CRYPTO_MAC_HMAC_SHA512_256_Final, **832**
 CRYPTO_MAC_HMAC_SHA512_256_Final_256, **833**
 CRYPTO_MAC_HMAC_SHA512_256_Init, **830**
 CRYPTO_MAC_HMAC_SHA512_256_InitEx, **831**
 CRYPTO_MAC_HMAC_SHA512_256_Kill, **834**
 CRYPTO_MAC_HMAC_SHA512_Add, **791**
 CRYPTO_MAC_HMAC_SHA512_Final, **792**
 CRYPTO_MAC_HMAC_SHA512_Final_512, **793**
 CRYPTO_MAC_HMAC_SHA512_Init, **794**
 CRYPTO_MAC_HMAC_SHA512_Kill, **795**
 CRYPTO_MAC_HMAC_SM3_Add, **914**
 CRYPTO_MAC_HMAC_SM3_Final, **915**
 CRYPTO_MAC_HMAC_SM3_Final_256, **916**
 CRYPTO_MAC_HMAC_SM3_Init, **917**
 CRYPTO_MAC_HMAC_SM3_InitEx, **918**
 CRYPTO_MAC_HMAC_SM3_Kill, **919**
 CRYPTO_MAC_MICHAEL_Add, **1130**
 CRYPTO_MAC_MICHAEL_Final, **1131**
 CRYPTO_MAC_MICHAEL_Final_64, **1132**
 CRYPTO_MAC_MICHAEL_Init, **1133**
 CRYPTO_MAC_MICHAEL_InitEx, **1134**
 CRYPTO_MAC_MICHAEL_Kill, **1135**
 CRYPTO_MAC_POLY1305_AES_Add, **1045**
 CRYPTO_MAC_POLY1305_AES_Final, **1046**
 CRYPTO_MAC_POLY1305_AES_Final_128, **1047**
 CRYPTO_MAC_POLY1305_AES_InitEx, **1048**
 CRYPTO_MAC_POLY1305_AES_Kill, **1049**
 CRYPTO_MAC_POLY1305_ARIA_Add, **1081**
 CRYPTO_MAC_POLY1305_ARIA_Final, **1082**
 CRYPTO_MAC_POLY1305_ARIA_Final_128, **1083**
 CRYPTO_MAC_POLY1305_ARIA_InitEx, **1084**
 CRYPTO_MAC_POLY1305_ARIA_Kill, **1085**
 CRYPTO_MAC_POLY1305_CAMELLIA_Add, **1098**
 CRYPTO_MAC_POLY1305_CAMELLIA_Final, **1099**
 CRYPTO_MAC_POLY1305_CAMELLIA_Final_128, **1100**
 CRYPTO_MAC_POLY1305_CAMELLIA_InitEx, **1101**
 CRYPTO_MAC_POLY1305_CAMELLIA_Kill, **1102**
 CRYPTO_MAC_POLY1305_SEED_Add, **1064**
 CRYPTO_MAC_POLY1305_SEED_Final, **1065**
 CRYPTO_MAC_POLY1305_SEED_Final_128, **1066**
 CRYPTO_MAC_POLY1305_SEED_InitEx, **1067**
 CRYPTO_MAC_POLY1305_SEED_Kill, **1068**
 CRYPTO_MAC_POLY1305_TWOFISH_Add, **1115**
 CRYPTO_MAC_POLY1305_TWOFISH_Final, **1116**
 CRYPTO_MAC_POLY1305_TWOFISH_Final_128, **1117**
 CRYPTO_MAC_POLY1305_TWOFISH_InitEx, **1118**
 CRYPTO_MAC_POLY1305_TWOFISH_Kill, **1119**
 CRYPTO_MAC_XCBC_AES_Add, **932**
 CRYPTO_MAC_XCBC_AES_Final, **933**

CRYPTO_MAC_XCBC_AES_Final_128, **934**
 CRYPTO_MAC_XCBC_AES_Init, **935**
 CRYPTO_MAC_XCBC_AES_InitEx, **936**
 CRYPTO_MAC_XCBC_AES_Kill, **937**
 CRYPTO_MAC_XCBC_ARIA_Add, **970**
 CRYPTO_MAC_XCBC_ARIA_Final, **971**
 CRYPTO_MAC_XCBC_ARIA_Final_128, **972**
 CRYPTO_MAC_XCBC_ARIA_Init, **973**
 CRYPTO_MAC_XCBC_ARIA_InitEx, **974**
 CRYPTO_MAC_XCBC_ARIA_Kill, **975**
 CRYPTO_MAC_XCBC_CAMELLIA_Add, **988**
 CRYPTO_MAC_XCBC_CAMELLIA_Final, **989**
 CRYPTO_MAC_XCBC_CAMELLIA_Final_128, **990**
 CRYPTO_MAC_XCBC_CAMELLIA_Init, **991**
 CRYPTO_MAC_XCBC_CAMELLIA_InitEx, **992**
 CRYPTO_MAC_XCBC_CAMELLIA_Kill, **993**
 CRYPTO_MAC_XCBC_SEED_Add, **952**
 CRYPTO_MAC_XCBC_SEED_Final, **953**
 CRYPTO_MAC_XCBC_SEED_Final_128, **954**
 CRYPTO_MAC_XCBC_SEED_Init, **955**
 CRYPTO_MAC_XCBC_SEED_InitEx, **956**
 CRYPTO_MAC_XCBC_SEED_Kill, **957**
 CRYPTO_MAC_XCBC_TWOFISH_Add, **1006**
 CRYPTO_MAC_XCBC_TWOFISH_Final, **1007**
 CRYPTO_MAC_XCBC_TWOFISH_Final_128, **1008**
 CRYPTO_MAC_XCBC_TWOFISH_Init, **1009**
 CRYPTO_MAC_XCBC_TWOFISH_InitEx, **1010**
 CRYPTO_MAC_XCBC_TWOFISH_Kill, **1011**
 CRYPTO_MD5_Add, **95**
 CRYPTO_MD5_Calc, **96**
 CRYPTO_MD5_Calc_160, **97**
 CRYPTO_MD5_Final, **98**
 CRYPTO_MD5_Final_160, **99**
 CRYPTO_MD5_Get, **100**
 CRYPTO_MD5_Init, **101**
 CRYPTO_MD5_Install, **102**, 2649, 2651, 2652, 2655, 2658, 2662, 2663, 2665
 CRYPTO_MD5_IsInstalled, **103**
 CRYPTO_MD5_Kill, **104**
 CRYPTO_MD5_QueryInstall, **105**, 115, 2771
 CRYPTO_MD5_RFC1321_SelfTest, **113**
 CRYPTO_MICHAEL_802v11_SelfTest, **1137**
 CRYPTO_MICHAEL_Add, **1121**
 CRYPTO_MICHAEL_Calc, **1122**
 CRYPTO_MICHAEL_Calc_64, **1123**
 CRYPTO_MICHAEL_Final, **1124**
 CRYPTO_MICHAEL_Final_64, **1125**
 CRYPTO_MICHAEL_Init, **1126**
 CRYPTO_MICHAEL_Init_64, **1127**
 CRYPTO_MICHAEL_Kill, **1128**
 CRYPTO_MPI_2Exp, **2537**
 CRYPTO_MPI_2ExpMinusOne, **2538**
 CRYPTO_MPI_Abs, **2486**
 CRYPTO_MPI_Add, **2487**, 2792, 2793
 CRYPTO_MPI_AddEx, **2488**
 CRYPTO_MPI_AddSmall, **2489**
 CRYPTO_MPI_AddUnsigned, **2490**
 CRYPTO_MPI_Assign, **2456**, 2791, 2791, 2792, 2792
 CRYPTO_MPI_AssignInt, **2457**
 CRYPTO_MPI_AssignU32, **2458**
 CRYPTO_MPI_AssignU64, **2459**
 CRYPTO_MPI_AssignUnsigned, **2460**
 CRYPTO_MPI_BitCount, 2248, 2249, 2390, **2579**, 2790, 2792, 2793, 2799
 CRYPTO_MPI_BytCount, **2580**
 CRYPTO_MPI_BytCount_ASNI, **2581**
 CRYPTO_MPI_CeilDiv, **2519**
 CRYPTO_MPI_Clear, **2449**
 CRYPTO_MPI_ClrBit, **2584**
 CRYPTO_MPI_Compare, **2465**
 CRYPTO_MPI_Dec, **2491**, 2791, 2791
 CRYPTO_MPI_Div, **2520**
 CRYPTO_MPI_Div2, **2521**
 CRYPTO_MPI_DivMod, **2522**
 CRYPTO_MPI_DivUnsigned, **2523**
 CRYPTO_MPI_Equate, **2461**
 CRYPTO_MPI_Evict, **2450**
 CRYPTO_MPI_Exchange, **2462**

CRYPTO_MPI_Exp, **2539**
 CRYPTO_MPI_ExtractBits, **2582**
 CRYPTO_MPI_FormatDecimal, **2609**
 CRYPTO_MPI_FormatHex, **2610**
 CRYPTO_MPI_GCD, **2598**
 CRYPTO_MPI_GCD_Binary, **2599**
 CRYPTO_MPI_GCD_Euclid, **2600**
 CRYPTO_MPI_GCD_Lehmer, **2601**
 CRYPTO_MPI_Inc, **2492**
 CRYPTO_MPI_Init, **2451**, 2790, 2791, 2791, 2791, 2791, 2792, 2792, 2792, 2799
 CRYPTO_MPI_IsCoprime, **2621**
 CRYPTO_MPI_IsEqual, 2390, **2466**
 CRYPTO_MPI_IsEven, **2467**
 CRYPTO_MPI_IsFermatProbablePrime, **2622**
 CRYPTO_MPI_IsGreater, **2468**
 CRYPTO_MPI_IsGreaterEqual, **2469**
 CRYPTO_MPI_IsGreaterZero, **2470**
 CRYPTO_MPI_IsLess, **2471**
 CRYPTO_MPI_IsLessEqual, **2472**
 CRYPTO_MPI_IsMRProbablePrime, **2623**
 CRYPTO_MPI_IsMRProbablePrimeEx, **2624**
 CRYPTO_MPI_IsNegative, **2473**, 2792
 CRYPTO_MPI_IsNonzero, **2474**
 CRYPTO_MPI_IsNotEqual, **2475**
 CRYPTO_MPI_IsOdd, **2476**
 CRYPTO_MPI_IsOne, **2477**
 CRYPTO_MPI_IsPositive, **2478**
 CRYPTO_MPI_IsProbableSafePrime, **2625**
 CRYPTO_MPI_IsProvableSmallPrime, **2626**
 CRYPTO_MPI_IsReadOnly, **2479**
 CRYPTO_MPI_IsReadWrite, **2480**
 CRYPTO_MPI_IsZero, **2481**
 CRYPTO_MPI_Kill, **2452**, 2791, 2791, 2791, 2791, 2793, 2793, 2793, 2799
 CRYPTO_MPI_LCM, **2602**, 2791
 CRYPTO_MPI_LimbsRequired, **2594**
 CRYPTO_MPI_Load, **2611**
 CRYPTO_MPI_LoadBits, **2612**
 CRYPTO_MPI_LoadBytes, 1962, 1962, 1963, 1963, 1963, 1963, 1963, 1990, 1990, 1991, 1991, 1991, 1991, 1991, 2169, 2170, 2170, 2170, 2170, **2613**
 CRYPTO_MPI_LoadBytesLE, **2614**
 CRYPTO_MPI_LoadDecimal, **2615**
 CRYPTO_MPI_LoadHex, **2616**, 2790, 2792, 2799
 CRYPTO_MPI_LoadText, **2617**
 CRYPTO_MPI_LSB, **2591**
 CRYPTO_MPI_Max, **2482**
 CRYPTO_MPI_Min, **2483**
 CRYPTO_MPI_Mod, **2524**, 2790, 2792
 CRYPTO_MPI_ModAdd, **2493**
 CRYPTO_MPI_ModAddEx, **2494**
 CRYPTO_MPI_ModDec, **2495**
 CRYPTO_MPI_ModDiv, **2525**
 CRYPTO_MPI_ModDiv2, **2526**
 CRYPTO_MPI_ModEx, **2527**
 CRYPTO_MPI_ModExp, **2540**, 2789
 CRYPTO_MPI_ModExp2Pow, **2541**
 CRYPTO_MPI_ModExp_Barrett_2b_FW, **2556**, 2789
 CRYPTO_MPI_ModExp_Barrett_2b_RM, **2561**, 2789
 CRYPTO_MPI_ModExp_Barrett_3b_FW, **2557**, 2789
 CRYPTO_MPI_ModExp_Barrett_3b_RM, **2562**, 2789
 CRYPTO_MPI_ModExp_Barrett_4b_FW, **2558**, 2789
 CRYPTO_MPI_ModExp_Barrett_4b_RM, **2563**, 2789
 CRYPTO_MPI_ModExp_Barrett_5b_FW, **2559**, 2789
 CRYPTO_MPI_ModExp_Barrett_5b_RM, **2564**, 2789
 CRYPTO_MPI_ModExp_Barrett_6b_FW, **2560**, 2789
 CRYPTO_MPI_ModExp_Barrett_6b_RM, **2565**, 2789
 CRYPTO_MPI_ModExp_Barrett_Fast, **2554**, 2789
 CRYPTO_MPI_ModExp_Barrett_Ladder, **2555**, 2789
 CRYPTO_MPI_ModExp_Basic_2b_FW, **2544**, 2788
 CRYPTO_MPI_ModExp_Basic_2b_RM, **2549**, 2788
 CRYPTO_MPI_ModExp_Basic_3b_FW, **2545**, 2788
 CRYPTO_MPI_ModExp_Basic_3b_RM, **2550**, 2788
 CRYPTO_MPI_ModExp_Basic_4b_FW, **2546**, 2788
 CRYPTO_MPI_ModExp_Basic_4b_RM, **2551**, 2788
 CRYPTO_MPI_ModExp_Basic_5b_FW, **2547**, 2788
 CRYPTO_MPI_ModExp_Basic_5b_RM, **2552**, 2789
 CRYPTO_MPI_ModExp_Basic_6b_FW, **2548**, 2788

CRYPTO_MPI_ModExp_Basic_6b_RM, **2553**, 2789
 CRYPTO_MPI_ModExp_Basic_Fast, **2542**, 2653, 2653, 2658, 2658, 2662, 2662, 2665, 2665, 2788
 CRYPTO_MPI_ModExp_Basic_Ladder, **2543**, 2788
 CRYPTO_MPI_ModExp_Montgomery_2b_FW, **2568**, 2789
 CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO, **2638**
 CRYPTO_MPI_ModExp_Montgomery_2b_RM, **2573**, 2789
 CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO, **2643**
 CRYPTO_MPI_ModExp_Montgomery_3b_FW, **2569**, 2789
 CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO, **2639**
 CRYPTO_MPI_ModExp_Montgomery_3b_RM, **2574**, 2789
 CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO, **2644**
 CRYPTO_MPI_ModExp_Montgomery_4b_FW, **2570**, 2789
 CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO, **2640**
 CRYPTO_MPI_ModExp_Montgomery_4b_RM, **2575**, 2789
 CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO, **2645**
 CRYPTO_MPI_ModExp_Montgomery_5b_FW, **2571**, 2789
 CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO, **2641**
 CRYPTO_MPI_ModExp_Montgomery_5b_RM, **2576**, 2789
 CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO, **2646**
 CRYPTO_MPI_ModExp_Montgomery_6b_FW, **2572**, 2789
 CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO, **2642**
 CRYPTO_MPI_ModExp_Montgomery_6b_RM, **2577**, 2789
 CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO, **2647**
 CRYPTO_MPI_ModExp_Montgomery_Fast, **2566**, 2789
 CRYPTO_MPI_ModExp_Montgomery_Ladder, **2567**, 2789
 CRYPTO_MPI_ModInc, **2496**
 CRYPTO_MPI_ModInv, **2528**
 CRYPTO_MPI_ModInvEx, **2529**, 2791
 CRYPTO_MPI_ModMul, **2508**, 2792
 CRYPTO_MPI_ModMul2, **2510**
 CRYPTO_MPI_ModMulEx, **2509**
 CRYPTO_MPI_ModNeg, **2497**
 CRYPTO_MPI_ModRevSub, **2500**
 CRYPTO_MPI_ModSqrt, **2530**
 CRYPTO_MPI_ModSquare, **2506**
 CRYPTO_MPI_ModSquareEx, **2507**
 CRYPTO_MPI_ModSub, **2498**
 CRYPTO_MPI_ModSubEx, **2499**
 CRYPTO_MPI_ModUnsigned, **2531**
 CRYPTO_MPI_Move, **2463**, 2792
 CRYPTO_MPI_MSB, **2592**
 CRYPTO_MPI_Mul, **2511**, 2793
 CRYPTO_MPI_Mul2, **2512**
 CRYPTO_MPI_MulEx, **2513**
 CRYPTO_MPI_MulUnsigned, **2514**
 CRYPTO_MPI_Neg, **2501**
 CRYPTO_MPI_NonzeroRandom, **2604**
 CRYPTO_MPI_NonzeroRandomEx, **2605**
 CRYPTO_MPI_P1363_CalcMRTrials, **2627**
 CRYPTO_MPI_QuerySmallPrimeFactor, **2628**
 CRYPTO_MPI_Random, **2606**
 CRYPTO_MPI_RandomBits, **2607**
 CRYPTO_MPI_Rdbit, **2585**
 CRYPTO_MPI_RdBits, **2586**
 CRYPTO_MPI_RdByte, **2587**
 CRYPTO_MPI_Reserve, **2453**
 CRYPTO_MPI_RevDiv, **2532**
 CRYPTO_MPI_RevSub, **2502**
 CRYPTO_MPI_SetBit, **2588**
 CRYPTO_MPI_SetChunkSize, 2790, 2792
 CRYPTO_MPI_SetPrivateModExp, 2653, 2658, 2662, 2665
 CRYPTO_MPI_SetPublicModExp, 2653, 2658, 2662, 2665
 CRYPTO_MPI_Sgn, **2484**
 CRYPTO_MPI_ShiftLeft, **2515**
 CRYPTO_MPI_ShiftRight, **2533**
 CRYPTO_MPI_Shrink, **2454**
 CRYPTO_MPI_Sqrt, **2534**
 CRYPTO_MPI_Square, **2516**
 CRYPTO_MPI_SquareEx, **2517**
 CRYPTO_MPI_StoreBytes, 2169, 2169, **2618**
 CRYPTO_MPI_StoreBytesLE, **2619**
 CRYPTO_MPI_Sub, **2503**, 2792
 CRYPTO_MPI_SubUnsigned, **2504**
 CRYPTO_MPI_TrimBits, **2589**, 2799
 CRYPTO_MPI_TrimLimbs, **2590**
 CRYPTO_MPI_Unsigned, **2593**

CRYPTO_MPI_WrBit, **2583**
 CRYPTO_MPI_Xor, **2596**
 CRYPTO_OS_Claim, **33**, 36, 39
 CRYPTO_OS_Init, **32**, 37, 40
 CRYPTO_OS_Request, **34**, 37, 39
 CRYPTO_OS_Unclaim, **35**, 37, 40
 CRYPTO_PBKDF2_HMAC_SHA1_Calc, **1917**
 CRYPTO_PBKDF2_HMAC_SHA224_Calc, **1918**
 CRYPTO_PBKDF2_HMAC_SHA256_Calc, **1919**
 CRYPTO_PBKDF2_HMAC_SHA384_Calc, **1920**
 CRYPTO_PBKDF2_HMAC_SHA512_224_Calc, **1922**
 CRYPTO_PBKDF2_HMAC_SHA512_256_Calc, **1923**
 CRYPTO_PBKDF2_HMAC_SHA512_Calc, **1921**
 CRYPTO_PBKDF2_HMAC_SM3_Calc, **1924**
 CRYPTO_PBKDF2_SelfTest, **1926**
 CRYPTO_POLY1305_Add, **1022**
 CRYPTO_POLY1305_AES_Add, **1034**
 CRYPTO_POLY1305_AES_Bernstein_SelfTest, **1051**
 CRYPTO_POLY1305_AES_Calc, **1035**
 CRYPTO_POLY1305_AES_Calc_128, **1036**
 CRYPTO_POLY1305_AES_Clamp, **1037**
 CRYPTO_POLY1305_AES_Final, **1038**
 CRYPTO_POLY1305_AES_Final_128, **1039**
 CRYPTO_POLY1305_AES_InitEx_256_128, **1040**
 CRYPTO_POLY1305_AES_Kill, **1041**
 CRYPTO_POLY1305_AES_Verify, **1042**
 CRYPTO_POLY1305_AES_Verify_128, **1043**
 CRYPTO_POLY1305_ARIA_Add, **1070**
 CRYPTO_POLY1305_ARIA_Calc, **1071**
 CRYPTO_POLY1305_ARIA_Calc_128, **1072**
 CRYPTO_POLY1305_ARIA_Clamp, **1073**
 CRYPTO_POLY1305_ARIA_Final, **1074**
 CRYPTO_POLY1305_ARIA_Final_128, **1075**
 CRYPTO_POLY1305_ARIA_InitEx_256_128, **1076**
 CRYPTO_POLY1305_ARIA_Kill, **1077**
 CRYPTO_POLY1305_ARIA_Verify, **1078**
 CRYPTO_POLY1305_ARIA_Verify_128, **1079**
 CRYPTO_POLY1305_Bernstein_SelfTest, **1032**
 CRYPTO_POLY1305_Calc, **1023**
 CRYPTO_POLY1305_Calc_128, **1024**
 CRYPTO_POLY1305_CAMELLIA_Add, **1087**
 CRYPTO_POLY1305_CAMELLIA_Calc, **1088**
 CRYPTO_POLY1305_CAMELLIA_Calc_128, **1089**
 CRYPTO_POLY1305_CAMELLIA_Clamp, **1090**
 CRYPTO_POLY1305_CAMELLIA_Final, **1091**
 CRYPTO_POLY1305_CAMELLIA_Final_128, **1092**
 CRYPTO_POLY1305_CAMELLIA_InitEx_256_128, **1093**
 CRYPTO_POLY1305_CAMELLIA_Kill, **1094**
 CRYPTO_POLY1305_CAMELLIA_Verify, **1095**
 CRYPTO_POLY1305_CAMELLIA_Verify_128, **1096**
 CRYPTO_POLY1305_Clamp, **1030**
 CRYPTO_POLY1305_Final, **1025**
 CRYPTO_POLY1305_Final_128, **1026**
 CRYPTO_POLY1305_Init, **1027**
 CRYPTO_POLY1305_Init_256, **1028**
 CRYPTO_POLY1305_Kill, **1029**
 CRYPTO_POLY1305_SEED_Add, **1053**
 CRYPTO_POLY1305_SEED_Calc, **1054**
 CRYPTO_POLY1305_SEED_Calc_128, **1055**
 CRYPTO_POLY1305_SEED_Clamp, **1056**
 CRYPTO_POLY1305_SEED_Final, **1057**
 CRYPTO_POLY1305_SEED_Final_128, **1058**
 CRYPTO_POLY1305_SEED_InitEx_256_128, **1059**
 CRYPTO_POLY1305_SEED_Kill, **1060**
 CRYPTO_POLY1305_SEED_Verify, **1061**
 CRYPTO_POLY1305_SEED_Verify_128, **1062**
 CRYPTO_POLY1305_TWOFISH_Add, **1104**
 CRYPTO_POLY1305_TWOFISH_Calc, **1105**
 CRYPTO_POLY1305_TWOFISH_Calc_128, **1106**
 CRYPTO_POLY1305_TWOFISH_Clamp, **1107**
 CRYPTO_POLY1305_TWOFISH_Final, **1108**
 CRYPTO_POLY1305_TWOFISH_Final_128, **1109**
 CRYPTO_POLY1305_TWOFISH_InitEx_256_128, **1110**
 CRYPTO_POLY1305_TWOFISH_Kill, **1111**
 CRYPTO_POLY1305_TWOFISH_Verify, **1112**
 CRYPTO_POLY1305_TWOFISH_Verify_128, **1113**

CRYPTO_PRESENT_CBC_Decrypt, **1618**
 CRYPTO_PRESENT_CBC_Encrypt, **1617**
 CRYPTO_PRESENT_CHES2007_SelfTest, **1645**
 CRYPTO_PRESENT_CTR_Decrypt, **1622**
 CRYPTO_PRESENT_CTR_Encrypt, **1621**
 CRYPTO_PRESENT_Decrypt, **1614**
 CRYPTO_PRESENT_ECB_Decrypt, **1616**
 CRYPTO_PRESENT_ECB_Encrypt, **1615**
 CRYPTO_PRESENT_Encrypt, **1613**
 CRYPTO_PRESENT_InitDecrypt, **1611**
 CRYPTO_PRESENT_InitEncrypt, **1610**
 CRYPTO_PRESENT_Install, **1607**
 CRYPTO_PRESENT_IsInstalled, **1608**
 CRYPTO_PRESENT_Kill, **1612**
 CRYPTO_PRESENT_OFB_Decrypt, **1620**
 CRYPTO_PRESENT_OFB_Encrypt, **1619**
 CRYPTO_PRESENT_QueryInstall, **1609**
 CRYPTO_PRIME_FindCoprime, **2631**
 CRYPTO_PRIME_FindPrime, **2629**
 CRYPTO_PRIME_FindPrimeFrom, **2630**
 CRYPTO_RC4_Decrypt, **1649**
 CRYPTO_RC4_Encrypt, **1648**
 CRYPTO_RC4_Prepares, **1650**
 CRYPTO_RIPEMD160_Add, **120**
 CRYPTO_RIPEMD160_Bosselaers_SelfTest, **138**
 CRYPTO_RIPEMD160_Calc, **121**
 CRYPTO_RIPEMD160_Calc_160, **122**
 CRYPTO_RIPEMD160_Final, **123**
 CRYPTO_RIPEMD160_Final_160, **124**
 CRYPTO_RIPEMD160_Get, **125**
 CRYPTO_RIPEMD160_Init, **126**
 CRYPTO_RIPEMD160_Install, **127**, 2649, 2653, 2655, 2658, 2662, 2665
 CRYPTO_RIPEMD160_IsInstalled, **128**
 CRYPTO_RIPEMD160_Kill, **129**
 CRYPTO_RIPEMD160_QueryInstall, **130**, 140
 CRYPTO_RNG_Install, **2391**
 CRYPTO_RNG_InstallEx, **2653**, 2656, 2658, 2662, 2665
 CRYPTO_RNG_QueryInstallEx, **2804**
 CRYPTO_RSA_CalcDecryptExponent, **1982**
 CRYPTO_RSA_ConstructKeys, **1983**
 CRYPTO_RSA_Decrypt, **1963**, **1977**
 CRYPTO_RSA_DecryptMPI, **1978**
 CRYPTO_RSA_DecryptMPIOnCRT, **1979**
 CRYPTO_RSA_DecryptMPIToMPI, **1980**
 CRYPTO_RSA_Encrypt, **1962**, **1973**
 CRYPTO_RSA_EncryptMPI, **1974**
 CRYPTO_RSA_EncryptMPIToMPI, **1975**
 CRYPTO_RSA_FIPS186_GenKeys, **1994**
 CRYPTO_RSA_FIPS186_GenPrime, **1995**
 CRYPTO_RSA_FIPS186_GenPrimePair, **1996**
 CRYPTO_RSA_FIPS186_ValidateParaSize, **1997**
 CRYPTO_RSA_InitPrivateKey, **1963**, **1968**, 1991
 CRYPTO_RSA_InitPublicKey, **1962**, **1969**, 1990
 CRYPTO_RSA_IsConsistentPair, **1984**
 CRYPTO_RSA_KillPrivateKey, **1970**
 CRYPTO_RSA_KillPublicKey, **1971**
 CRYPTO_RSA_ModulusBits, **1985**
 CRYPTO_RSA_ModulusBytes, **1986**
 CRYPTO_RSA_ModulusBytes_ASN1, **1987**
 CRYPTO_RSA_P1363_GenKeys, **1993**
 CRYPTO_RSA_PKCS1_Unwrap, **2046**
 CRYPTO_RSA_RecoverModulus, **1988**
 CRYPTO_RSA_SEGGER_SelfTest, **2107**
 CRYPTO_RSA_SHA1_KeyGen_CAVS_SelfTest, **2100**
 CRYPTO_RSA_SHA224_KeyGen_CAVS_SelfTest, **2101**
 CRYPTO_RSA_SHA256_KeyGen_CAVS_SelfTest, **2102**
 CRYPTO_RSA_SHA384_KeyGen_CAVS_SelfTest, **2103**
 CRYPTO_RSA_SHA512_224_KeyGen_CAVS_SelfTest, **2104**
 CRYPTO_RSA_SHA512_256_KeyGen_CAVS_SelfTest, **2105**
 CRYPTO_RSA_SHA512_KeyGen_CAVS_SelfTest, **2106**
 CRYPTO_RSAES_OAEP_EMC_SelfTest, **2328**
 CRYPTO_RSAES_OAEP_KDF1_SHA1_Decrypt, **2316**
 CRYPTO_RSAES_OAEP_KDF1_SHA1_Encrypt, **2305**
 CRYPTO_RSAES_OAEP_KDF1_SHA224_Decrypt, **2317**
 CRYPTO_RSAES_OAEP_KDF1_SHA224_Encrypt, **2306**
 CRYPTO_RSAES_OAEP_KDF1_SHA256_Decrypt, **2318**

CRYPTO_RSAES_OAEP_KDF1_SHA256_Encrypt, **2307**
 CRYPTO_RSAES_OAEP_KDF1_SHA384_Decrypt, **2319**
 CRYPTO_RSAES_OAEP_KDF1_SHA384_Encrypt, **2308**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Decrypt, **2323**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_224_Encrypt, **2312**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Decrypt, **2324**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_256_Encrypt, **2313**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Decrypt, **2325**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_384_Encrypt, **2314**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Decrypt, **2326**
 CRYPTO_RSAES_OAEP_KDF1_SHA3_512_Encrypt, **2315**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Decrypt, **2321**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_224_Encrypt, **2310**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Decrypt, **2322**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_256_Encrypt, **2311**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_Decrypt, **2320**
 CRYPTO_RSAES_OAEP_KDF1_SHA512_Encrypt, **2309**
 CRYPTO_RSAES_PKCS1_Encrypt, **2330**
 CRYPTO_RSASSA_PKCS1_SHA1_Sign, 1991, **1999**
 CRYPTO_RSASSA_PKCS1_SHA1_SignDigest, **2023**
 CRYPTO_RSASSA_PKCS1_SHA1_Verify, 1990, **2000**
 CRYPTO_RSASSA_PKCS1_SHA1_VerifyDigest, **2024**
 CRYPTO_RSASSA_PKCS1_SHA224_Sign, **2001**
 CRYPTO_RSASSA_PKCS1_SHA224_SignDigest, **2025**
 CRYPTO_RSASSA_PKCS1_SHA224_Verify, **2002**
 CRYPTO_RSASSA_PKCS1_SHA224_VerifyDigest, **2026**
 CRYPTO_RSASSA_PKCS1_SHA256_Sign, **2003**
 CRYPTO_RSASSA_PKCS1_SHA256_SignDigest, **2027**
 CRYPTO_RSASSA_PKCS1_SHA256_Verify, **2004**
 CRYPTO_RSASSA_PKCS1_SHA256_VerifyDigest, **2028**
 CRYPTO_RSASSA_PKCS1_SHA384_Sign, **2005**
 CRYPTO_RSASSA_PKCS1_SHA384_SignDigest, **2029**
 CRYPTO_RSASSA_PKCS1_SHA384_Verify, **2006**
 CRYPTO_RSASSA_PKCS1_SHA384_VerifyDigest, **2030**
 CRYPTO_RSASSA_PKCS1_SHA3_224_Sign, **2013**
 CRYPTO_RSASSA_PKCS1_SHA3_224_SignDigest, **2037**
 CRYPTO_RSASSA_PKCS1_SHA3_224_Verify, **2014**
 CRYPTO_RSASSA_PKCS1_SHA3_224_VerifyDigest, **2038**
 CRYPTO_RSASSA_PKCS1_SHA3_256_Sign, **2015**
 CRYPTO_RSASSA_PKCS1_SHA3_256_SignDigest, **2039**
 CRYPTO_RSASSA_PKCS1_SHA3_256_Verify, **2016**
 CRYPTO_RSASSA_PKCS1_SHA3_256_VerifyDigest, **2040**
 CRYPTO_RSASSA_PKCS1_SHA3_384_Sign, **2017**
 CRYPTO_RSASSA_PKCS1_SHA3_384_SignDigest, **2041**
 CRYPTO_RSASSA_PKCS1_SHA3_384_Verify, **2018**
 CRYPTO_RSASSA_PKCS1_SHA3_384_VerifyDigest, **2042**
 CRYPTO_RSASSA_PKCS1_SHA3_512_Sign, **2019**
 CRYPTO_RSASSA_PKCS1_SHA3_512_SignDigest, **2043**
 CRYPTO_RSASSA_PKCS1_SHA3_512_Verify, **2020**
 CRYPTO_RSASSA_PKCS1_SHA3_512_VerifyDigest, **2044**
 CRYPTO_RSASSA_PKCS1_SHA512_224_Sign, **2007**
 CRYPTO_RSASSA_PKCS1_SHA512_224_SignDigest, **2031**
 CRYPTO_RSASSA_PKCS1_SHA512_224_Verify, **2008**
 CRYPTO_RSASSA_PKCS1_SHA512_224_VerifyDigest, **2032**
 CRYPTO_RSASSA_PKCS1_SHA512_256_Sign, **2009**
 CRYPTO_RSASSA_PKCS1_SHA512_256_SignDigest, **2033**
 CRYPTO_RSASSA_PKCS1_SHA512_256_Verify, **2010**
 CRYPTO_RSASSA_PKCS1_SHA512_256_VerifyDigest, **2034**
 CRYPTO_RSASSA_PKCS1_SHA512_Sign, **2011**
 CRYPTO_RSASSA_PKCS1_SHA512_SignDigest, **2035**
 CRYPTO_RSASSA_PKCS1_SHA512_Verify, **2012**
 CRYPTO_RSASSA_PKCS1_SHA512_VerifyDigest, **2036**
 CRYPTO_RSASSA_PKCS1_Sign_CAVS_SelfTest, **2094**
 CRYPTO_RSASSA_PKCS1_Sign_EMCSelfTest, **2095**
 CRYPTO_RSASSA_PKCS1_SignDigest, **2045**
 CRYPTO_RSASSA_PKCS1_Verify_CAVS_SelfTest, **2096**
 CRYPTO_RSASSA_PSS_SHA1_Sign, **2048**
 CRYPTO_RSASSA_PSS_SHA1_SignDigest, **2071**
 CRYPTO_RSASSA_PSS_SHA1_Verify, **2049**
 CRYPTO_RSASSA_PSS_SHA1_VerifyDigest, **2072**
 CRYPTO_RSASSA_PSS_SHA224_Sign, **2050**
 CRYPTO_RSASSA_PSS_SHA224_SignDigest, **2073**
 CRYPTO_RSASSA_PSS_SHA224_Verify, **2051**
 CRYPTO_RSASSA_PSS_SHA224_VerifyDigest, **2074**
 CRYPTO_RSASSA_PSS_SHA256_Sign, **2052**
 CRYPTO_RSASSA_PSS_SHA256_SignDigest, **2075**

CRYPTO_RSASSA_PSS_SHA256_Verify, **2053**
 CRYPTO_RSASSA_PSS_SHA256_VerifyDigest, **2076**
 CRYPTO_RSASSA_PSS_SHA384_Sign, **2054**
 CRYPTO_RSASSA_PSS_SHA384_SignDigest, **2077**
 CRYPTO_RSASSA_PSS_SHA384_Verify, **2055**
 CRYPTO_RSASSA_PSS_SHA384_VerifyDigest, **2078**
 CRYPTO_RSASSA_PSS_SHA3_224_Sign, **2062**
 CRYPTO_RSASSA_PSS_SHA3_224_SignDigest, **2085**
 CRYPTO_RSASSA_PSS_SHA3_224_Verify, **2063**
 CRYPTO_RSASSA_PSS_SHA3_224_VerifyDigest, **2086**
 CRYPTO_RSASSA_PSS_SHA3_256_Sign, **2064**
 CRYPTO_RSASSA_PSS_SHA3_256_SignDigest, **2087**
 CRYPTO_RSASSA_PSS_SHA3_256_Verify, **2065**
 CRYPTO_RSASSA_PSS_SHA3_256_VerifyDigest, **2088**
 CRYPTO_RSASSA_PSS_SHA3_384_Sign, **2066**
 CRYPTO_RSASSA_PSS_SHA3_384_SignDigest, **2089**
 CRYPTO_RSASSA_PSS_SHA3_384_Verify, **2067**
 CRYPTO_RSASSA_PSS_SHA3_384_VerifyDigest, **2090**
 CRYPTO_RSASSA_PSS_SHA3_512_Sign, **2068**
 CRYPTO_RSASSA_PSS_SHA3_512_SignDigest, **2091**
 CRYPTO_RSASSA_PSS_SHA3_512_Verify, **2069**
 CRYPTO_RSASSA_PSS_SHA3_512_VerifyDigest, **2092**
 CRYPTO_RSASSA_PSS_SHA512_224_Sign, **2056**
 CRYPTO_RSASSA_PSS_SHA512_224_SignDigest, **2079**
 CRYPTO_RSASSA_PSS_SHA512_224_Verify, **2057**
 CRYPTO_RSASSA_PSS_SHA512_224_VerifyDigest, **2080**
 CRYPTO_RSASSA_PSS_SHA512_256_Sign, **2058**
 CRYPTO_RSASSA_PSS_SHA512_256_SignDigest, **2081**
 CRYPTO_RSASSA_PSS_SHA512_256_Verify, **2059**
 CRYPTO_RSASSA_PSS_SHA512_256_VerifyDigest, **2082**
 CRYPTO_RSASSA_PSS_SHA512_Sign, **2060**
 CRYPTO_RSASSA_PSS_SHA512_SignDigest, **2083**
 CRYPTO_RSASSA_PSS_SHA512_Verify, **2061**
 CRYPTO_RSASSA_PSS_SHA512_VerifyDigest, **2084**
 CRYPTO_RSASSA_PSS_Sign_CAVS_SelfTest, **2097**
 CRYPTO_RSASSA_PSS_Sign_EMU_SelfTest, **2098**
 CRYPTO_RSASSA_PSS_Verify_CAVS_SelfTest, **2099**
 CRYPTO_SEED_CBC_Decrypt, **1307**
 CRYPTO_SEED_CBC_Encrypt, **1306**
 CRYPTO_SEED_CCM_Decrypt, **1313**
 CRYPTO_SEED_CCM_Encrypt, **1312**
 CRYPTO_SEED_CTR_Decrypt, **1309**
 CRYPTO_SEED_CTR_Encrypt, **1308**
 CRYPTO_SEED_Decrypt, **1303**
 CRYPTO_SEED_ECB_Decrypt, **1305**
 CRYPTO_SEED_ECB_Encrypt, **1304**
 CRYPTO_SEED_Encrypt, **1302**
 CRYPTO_SEED_GCM_Decrypt, **1315**
 CRYPTO_SEED_GCM_Encrypt, **1314**
 CRYPTO_SEED_InitDecrypt, **1300**
 CRYPTO_SEED_InitEncrypt, **1299**
 CRYPTO_SEED_Install, **1296**, 2653, 2658, 2662, 2665
 CRYPTO_SEED_IsInstalled, **1297**
 CRYPTO_SEED_Kill, **1301**
 CRYPTO_SEED_OFB_Decrypt, **1311**
 CRYPTO_SEED_OFB_Encrypt, **1310**
 CRYPTO_SEED_QueryInstall, **1298**
 CRYPTO_SEED_RFC4269_SelfTest, **1344**
 CRYPTO_SHA1_Add, 48, 48, **146**
 CRYPTO_SHA1_Calc, **147**
 CRYPTO_SHA1_Calc_160, **148**
 CRYPTO_SHA1_CAVS_SelfTest, **164**
 CRYPTO_SHA1_Final, 48, **149**
 CRYPTO_SHA1_Final_160, **150**
 CRYPTO_SHA1_FIPS180_SelfTest, **165**
 CRYPTO_SHA1_Get, **151**
 CRYPTO_SHA1_Init, 48, **152**
 CRYPTO_SHA1_Install, **153**, 2637, 2649, 2651, 2652, 2655, 2656, 2657, 2658, 2662, 2663, 2665
 CRYPTO_SHA1_IsInstalled, **154**, 2805, 2805
 CRYPTO_SHA1_Kill, **155**
 CRYPTO_SHA1_QueryInstall, **156**, 168
 CRYPTO_SHA224_Add, **171**
 CRYPTO_SHA224_Calc, **172**
 CRYPTO_SHA224_Calc_224, **173**
 CRYPTO_SHA224_CAVS_SelfTest, **189**
 CRYPTO_SHA224_Final, **174**

CRYPTO_SHA224_Final_224, **175**
CRYPTO_SHA224_Get, **176**
CRYPTO_SHA224_Init, **177**
CRYPTO_SHA224_Install, **178**, 2649, 2651, 2652, 2658, 2662, 2663, 2665
CRYPTO_SHA224_IsInstalled, **179**, 2805, 2805
CRYPTO_SHA224_Kill, **180**
CRYPTO_SHA224_QueryInstall, **181**, 215, 2774
CRYPTO_SHA256_Add, **193**
CRYPTO_SHA256_Calc, **194**
CRYPTO_SHA256_Calc_256, **195**
CRYPTO_SHA256_CAVS_SelfTest, **211**
CRYPTO_SHA256_Final, **196**
CRYPTO_SHA256_Final_256, **197**
CRYPTO_SHA256_FIPS180_SelfTest, **212**
CRYPTO_SHA256_Get, **198**
CRYPTO_SHA256_Init, **199**
CRYPTO_SHA256_Install, **200**, 2649, 2651, 2652, 2655, 2656, 2657, 2658, 2662, 2663, 2665
CRYPTO_SHA256_IsInstalled, **201**, 2805, 2805
CRYPTO_SHA256_Kill, **202**
CRYPTO_SHA256_QueryInstall, **203**, 215, 2774
CRYPTO_SHA384_Add, **218**
CRYPTO_SHA384_Calc, **219**
CRYPTO_SHA384_Calc_384, **220**
CRYPTO_SHA384_CAVS_SelfTest, **233**
CRYPTO_SHA384_Final, **221**
CRYPTO_SHA384_Final_384, **222**
CRYPTO_SHA384_Get, **223**
CRYPTO_SHA384_Init, **224**
CRYPTO_SHA384_Kill, **225**
CRYPTO_SHA3_224_Add, **294**
CRYPTO_SHA3_224_Calc, **295**
CRYPTO_SHA3_224_Calc_224, **296**
CRYPTO_SHA3_224_CAVS_SelfTest, **312**
CRYPTO_SHA3_224_Final, **297**
CRYPTO_SHA3_224_Final_224, **298**
CRYPTO_SHA3_224_FIPS202_SelfTest, **313**
CRYPTO_SHA3_224_Get, **299**
CRYPTO_SHA3_224_Init, **300**
CRYPTO_SHA3_224_Install, **301**
CRYPTO_SHA3_224_IsInstalled, **302**
CRYPTO_SHA3_224_Kill, **303**
CRYPTO_SHA3_224_QueryInstall, **304**
CRYPTO_SHA3_256_Add, **316**
CRYPTO_SHA3_256_Calc, **317**
CRYPTO_SHA3_256_Calc_256, **318**
CRYPTO_SHA3_256_CAVS_SelfTest, **334**
CRYPTO_SHA3_256_Final, **319**
CRYPTO_SHA3_256_Final_256, **320**
CRYPTO_SHA3_256_FIPS202_SelfTest, **335**
CRYPTO_SHA3_256_Get, **321**
CRYPTO_SHA3_256_Init, **322**
CRYPTO_SHA3_256_Install, **323**
CRYPTO_SHA3_256_IsInstalled, **324**
CRYPTO_SHA3_256_Kill, **325**
CRYPTO_SHA3_256_QueryInstall, **326**
CRYPTO_SHA3_384_Add, **338**
CRYPTO_SHA3_384_Calc, **339**
CRYPTO_SHA3_384_Calc_384, **340**
CRYPTO_SHA3_384_CAVS_SelfTest, **356**
CRYPTO_SHA3_384_Final, **341**
CRYPTO_SHA3_384_Final_384, **342**
CRYPTO_SHA3_384_FIPS202_SelfTest, **357**
CRYPTO_SHA3_384_Get, **343**
CRYPTO_SHA3_384_Init, **344**
CRYPTO_SHA3_384_Install, **345**
CRYPTO_SHA3_384_IsInstalled, **346**
CRYPTO_SHA3_384_Kill, **347**
CRYPTO_SHA3_384_QueryInstall, **348**
CRYPTO_SHA3_512_Add, **360**
CRYPTO_SHA3_512_Calc, **361**
CRYPTO_SHA3_512_Calc_512, **362**
CRYPTO_SHA3_512_CAVS_SelfTest, **378**
CRYPTO_SHA3_512_Final, **363**
CRYPTO_SHA3_512_Final_512, **364**
CRYPTO_SHA3_512_FIPS202_SelfTest, **379**
CRYPTO_SHA3_512_Get, **365**

CRYPTO_SHA3_512_Init, **366**
CRYPTO_SHA3_512_Install, **367**
CRYPTO_SHA3_512_IsInstalled, **368**
CRYPTO_SHA3_512_Kill, **369**
CRYPTO_SHA3_512_QueryInstall, **370**
CRYPTO_SHA512_224_Add, **262**
CRYPTO_SHA512_224_Calc, **263**
CRYPTO_SHA512_224_Calc_224, **264**
CRYPTO_SHA512_224_Final, **265**
CRYPTO_SHA512_224_Final_224, **266**
CRYPTO_SHA512_224_Get, **267**
CRYPTO_SHA512_224_Init, **268**
CRYPTO_SHA512_224_Kill, **269**
CRYPTO_SHA512_256_Add, **278**
CRYPTO_SHA512_256_Calc, **279**
CRYPTO_SHA512_256_Calc_256, **280**
CRYPTO_SHA512_256_Final, **281**
CRYPTO_SHA512_256_Final_256, **282**
CRYPTO_SHA512_256_Get, **283**
CRYPTO_SHA512_256_Init, **284**
CRYPTO_SHA512_256_Kill, **285**
CRYPTO_SHA512_Add, **237**
CRYPTO_SHA512_Calc, **238**
CRYPTO_SHA512_Calc_512, **239**
CRYPTO_SHA512_CAVS_SelfTest, **255**
CRYPTO_SHA512_Final, **240**
CRYPTO_SHA512_Final_512, **241**
CRYPTO_SHA512_FIPS180_SelfTest, **256**
CRYPTO_SHA512_Get, **242**
CRYPTO_SHA512_Init, **243**
CRYPTO_SHA512_Install, **244**, 2649, 2653, 2655, 2658, 2662, 2665
CRYPTO_SHA512_IsInstalled, **245**, 2805, 2805
CRYPTO_SHA512_Kill, **246**
CRYPTO_SHA512_QueryInstall, **247**, 258, 2777
CRYPTO_SHAKE128_Add, **1929**
CRYPTO_SHAKE128_Calc, **1930**
CRYPTO_SHAKE128_CAVS_SelfTest, **1935**
CRYPTO_SHAKE128_Final, **1931**
CRYPTO_SHAKE128_Init, **1932**
CRYPTO_SHAKE128_Kill, **1933**
CRYPTO_SHAKE256_Add, **1937**
CRYPTO_SHAKE256_Calc, **1938**
CRYPTO_SHAKE256_CAVS_SelfTest, **1943**
CRYPTO_SHAKE256_Final, **1939**
CRYPTO_SHAKE256_Init, **1940**
CRYPTO_SHAKE256_Kill, **1941**
CRYPTO_SM3_Add, **383**
CRYPTO_SM3_Calc, **384**
CRYPTO_SM3_Calc_256, **385**
CRYPTO_SM3_Final, **386**
CRYPTO_SM3_Final_256, **387**
CRYPTO_SM3_GBT_SelfTest, **401**
CRYPTO_SM3_Get, **388**
CRYPTO_SM3_Init, **389**
CRYPTO_SM3_Install, **390**
CRYPTO_SM3_IsInstalled, **391**
CRYPTO_SM3_Kill, **392**
CRYPTO_SM3_QueryInstall, **393**
CRYPTO_TDES_CBC_CAVS_SelfTest, **1189**
CRYPTO_TDES_CBC_Decrypt, **1156**
CRYPTO_TDES_CBC_Encrypt, **1155**
CRYPTO_TDES_CheckParity, **1161**
CRYPTO_TDES_CorrectParity, **1162**
CRYPTO_TDES_CTR_Decrypt, **1160**
CRYPTO_TDES_CTR_Encrypt, **1159**
CRYPTO_TDES_Decrypt, **1152**
CRYPTO_TDES_ECB_CAVS_SelfTest, **1188**
CRYPTO_TDES_ECB_Decrypt, **1154**
CRYPTO_TDES_ECB_Encrypt, **1153**
CRYPTO_TDES_Encrypt, **1151**
CRYPTO_TDES_InitDecrypt, **1148**
CRYPTO_TDES_InitDecryptEx, **1149**
CRYPTO_TDES_InitEncrypt, **1146**
CRYPTO_TDES_InitEncryptEx, **1147**
CRYPTO_TDES_InsertParity, **1163**
CRYPTO_TDES_Install, **1143**, 2649, 2651, 2652, 2655, 2657, 2660, 2662, 2665

CRYPTO_TDES_IsInstalled, **1144**, 2805
 CRYPTO_TDES_Kill, **1150**
 CRYPTO_TDES_OFB_Decrypt, **1158**
 CRYPTO_TDES_OFB_Encrypt, **1157**
 CRYPTO_TDES_QueryInstall, **1145**, 2762
 CRYPTO_TLV_Accept, **2716**
 CRYPTO_TLV_AcceptStr, **2717**
 CRYPTO_TLV_Capture, **2718**
 CRYPTO_TLV_CaptureTo, **2719**
 CRYPTO_TLV_CaptureToNL, **2720**
 CRYPTO_TLV_CaptureValue, **2721**
 CRYPTO_TLV_CheckNull, **2722**
 CRYPTO_TLV_Close, **2723**
 CRYPTO_TLV_Copy, **2724**
 CRYPTO_TLV_EnsureBytes, **2725**
 CRYPTO_TLV_ForceClose, **2726**
 CRYPTO_TLV_GetNumUnread, **2727**
 CRYPTO_TLV_Init, **2728**
 CRYPTO_TLV_IsCompletelyRead, **2729**
 CRYPTO_TLV_IsValueEqual, **2730**
 CRYPTO_TLV_MatchValues, **2733**
 CRYPTO_TLV_MPI_LoadBytes, **2731**
 CRYPTO_TLV_MPI_LoadBytesLE, **2732**
 CRYPTO_TLV_ParseINTEGER, **2734**
 CRYPTO_TLV_ParseTagAndLength, **2735**
 CRYPTO_TLV_PeekTag, **2736**
 CRYPTO_TLV_PeekU8, **2737**
 CRYPTO_TLV_Prepare, **2738**
 CRYPTO_TLV_Read, **2739**
 CRYPTO_TLV_ReadU16, **2740**
 CRYPTO_TLV_ReadU24, **2741**
 CRYPTO_TLV_ReadU32, **2742**
 CRYPTO_TLV_ReadU8, **2743**
 CRYPTO_TLV_SkipBytes, **2744**
 CRYPTO_TLV_SkipINTEGER, **2745**
 CRYPTO_TLV_SkipNL, **2746**
 CRYPTO_TLV_Trim, **2747**
 CRYPTO_TWOFISH_CBC_Decrypt, **1566**
 CRYPTO_TWOFISH_CBC_Encrypt, **1565**
 CRYPTO_TWOFISH_CCM_Decrypt, **1572**
 CRYPTO_TWOFISH_CCM_Encrypt, **1571**
 CRYPTO_TWOFISH_CTR_Decrypt, **1570**
 CRYPTO_TWOFISH_CTR_Encrypt, **1569**
 CRYPTO_TWOFISH_Decrypt, **1562**
 CRYPTO_TWOFISH_ECB_Decrypt, **1564**
 CRYPTO_TWOFISH_ECB_Encrypt, **1563**
 CRYPTO_TWOFISH_Encrypt, **1561**
 CRYPTO_TWOFISH_GCM_Decrypt, **1574**
 CRYPTO_TWOFISH_GCM_Encrypt, **1573**
 CRYPTO_TWOFISH_InitDecrypt, **1559**
 CRYPTO_TWOFISH_InitEncrypt, **1558**
 CRYPTO_TWOFISH_Install, **1555**, 2653, 2658
 CRYPTO_TWOFISH_IsInstalled, **1556**
 CRYPTO_TWOFISH_Kill, **1560**
 CRYPTO_TWOFISH_OFB_Decrypt, **1568**
 CRYPTO_TWOFISH_OFB_Encrypt, **1567**
 CRYPTO_TWOFISH_QueryInstall, **1557**
 CRYPTO_TWOFISH_Schneier_SelfTest, **1603**
 CRYPTO_X9v63_KDF_BLAKE2B_Calc, **1878**
 CRYPTO_X9v63_KDF_BLAKE2B_CalcEx, **1879**
 CRYPTO_X9v63_KDF_BLAKE2S_Calc, **1880**
 CRYPTO_X9v63_KDF_BLAKE2S_CalcEx, **1881**
 CRYPTO_X9v63_KDF_SHA1_Calc, **1862**
 CRYPTO_X9v63_KDF_SHA1_CalcEx, **1863**
 CRYPTO_X9v63_KDF_SHA224_Calc, **1864**
 CRYPTO_X9v63_KDF_SHA224_CalcEx, **1865**
 CRYPTO_X9v63_KDF_SHA256_Calc, **1866**
 CRYPTO_X9v63_KDF_SHA256_CalcEx, **1867**
 CRYPTO_X9v63_KDF_SHA384_Calc, **1868**
 CRYPTO_X9v63_KDF_SHA384_CalcEx, **1869**
 CRYPTO_X9v63_KDF_SHA512_224_Calc, **1872**
 CRYPTO_X9v63_KDF_SHA512_224_CalcEx, **1873**
 CRYPTO_X9v63_KDF_SHA512_256_Calc, **1874**
 CRYPTO_X9v63_KDF_SHA512_256_CalcEx, **1875**
 CRYPTO_X9v63_KDF_SHA512_Calc, **1870**
 CRYPTO_X9v63_KDF_SHA512_CalcEx, **1871**

CRYPTO_X9v63_KDF_SM3_Calc, **1876**
CRYPTO_X9v63_KDF_SM3_CalcEx, **1877**
CRYPTO_XCBC_AES_Add, **922**
CRYPTO_XCBC_AES_Calc, **923**
CRYPTO_XCBC_AES_Calc_128, **925**
CRYPTO_XCBC_AES_Calc_96, **924**
CRYPTO_XCBC_AES_Final, **926**
CRYPTO_XCBC_AES_Final_128, **927**
CRYPTO_XCBC_AES_Init, **928**
CRYPTO_XCBC_AES_InitEx, **929**
CRYPTO_XCBC_AES_Kill, **930**
CRYPTO_XCBC_AES_RFC3566_SelfTest, **939**
CRYPTO_XCBC_ARIA_Add, **960**
CRYPTO_XCBC_ARIA_Calc, **961**
CRYPTO_XCBC_ARIA_Calc_128, **963**
CRYPTO_XCBC_ARIA_Calc_96, **962**
CRYPTO_XCBC_ARIA_Final, **964**
CRYPTO_XCBC_ARIA_Final_128, **965**
CRYPTO_XCBC_ARIA_Init, **966**
CRYPTO_XCBC_ARIA_InitEx, **967**
CRYPTO_XCBC_ARIA_Kill, **968**
CRYPTO_XCBC_CAMELLIA_Add, **978**
CRYPTO_XCBC_CAMELLIA_Calc, **979**
CRYPTO_XCBC_CAMELLIA_Calc_128, **981**
CRYPTO_XCBC_CAMELLIA_Calc_96, **980**
CRYPTO_XCBC_CAMELLIA_Final, **982**
CRYPTO_XCBC_CAMELLIA_Final_128, **983**
CRYPTO_XCBC_CAMELLIA_Init, **984**
CRYPTO_XCBC_CAMELLIA_InitEx, **985**
CRYPTO_XCBC_CAMELLIA_Kill, **986**
CRYPTO_XCBC_SEED_Add, **942**
CRYPTO_XCBC_SEED_Calc, **943**
CRYPTO_XCBC_SEED_Calc_128, **945**
CRYPTO_XCBC_SEED_Calc_96, **944**
CRYPTO_XCBC_SEED_Final, **946**
CRYPTO_XCBC_SEED_Final_128, **947**
CRYPTO_XCBC_SEED_Init, **948**
CRYPTO_XCBC_SEED_InitEx, **949**
CRYPTO_XCBC_SEED_Kill, **950**
CRYPTO_XCBC_TWOFISH_Add, **996**
CRYPTO_XCBC_TWOFISH_Calc, **997**
CRYPTO_XCBC_TWOFISH_Calc_128, **999**
CRYPTO_XCBC_TWOFISH_Calc_96, **998**
CRYPTO_XCBC_TWOFISH_Final, **1000**
CRYPTO_XCBC_TWOFISH_Final_128, **1001**
CRYPTO_XCBC_TWOFISH_Init, **1002**
CRYPTO_XCBC_TWOFISH_InitEx, **1003**
CRYPTO_XCBC_TWOFISH_Kill, **1004**
CRYPTO_XTS_AES_Decrypt, **1668**
CRYPTO_XTS_AES_Encrypt, **1667**
CRYPTO_XTS_ARIA_Decrypt, **1671**
CRYPTO_XTS_ARIA_Encrypt, **1670**
CRYPTO_XTS_CAMELLIA_Decrypt, **1674**
CRYPTO_XTS_CAMELLIA_Encrypt, **1673**
CRYPTO_XTS_SEED_Decrypt, **1677**
CRYPTO_XTS_SEED_Encrypt, **1676**
CRYPTO_XTS_TWOFISH_Decrypt, **1680**
CRYPTO_XTS_TWOFISH_Encrypt, **1679**

25.2 Subject index

- AES,
 - configuration,
 - compile-time, 1191
 - memory footprint, 1191
- ARIA,
 - configuration,
 - compile-time, 1346, 1605
 - memory footprint, 1346
- Blowfish,
 - configuration,
 - compile-time, 1526
 - memory footprint, 1526
- Camellia,
 - configuration,
 - compile-time, 1398
 - memory footprint, 1398
- CAST,
 - configuration,
 - compile-time, 1451
 - memory footprint, 1451
- Cipher API,
 - implementation,
 - `CRYPTO_CIPHER_AES_HW_Kinetis_CAU`, 2651
 - `CRYPTO_CIPHER_AES_HW_LPC_ROM`, 2654
 - `CRYPTO_CIPHER_AES_HW_RT10xx_DCP`, 2656
 - `CRYPTO_CIPHER_AES_HW_STM32_AES`, 2659
 - `CRYPTO_CIPHER_AES_HW_STM32_CRYP`, 2660
 - `CRYPTO_CIPHER_TDES_HW_Kinetis_CAU`, 2651
 - `CRYPTO_CIPHER_TDES_HW_STM32_CRYP`, 2660
- DES,
 - configuration,
 - compile-time, 1141
 - memory footprint, 1141
- Hash API,
 - implementation,
 - `CRYPTO_HASH_MD5_HW_Kinetis_CAU`, 2651
 - `CRYPTO_HASH_MD5_HW_STM32_HASH`, 2663
 - `CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO`, 2637
 - `CRYPTO_HASH_SHA1_HW_Kinetis_CAU`, 2651
 - `CRYPTO_HASH_SHA1_HW_RT10xx_DCP`, 2656
 - `CRYPTO_HASH_SHA1_HW_STM32_HASH`, 2663
 - `CRYPTO_HASH_SHA224_HW_Kinetis_CAU`, 2651
 - `CRYPTO_HASH_SHA224_HW_STM32_HASH`, 2663
 - `CRYPTO_HASH_SHA256_HW_Kinetis_CAU`, 2651
 - `CRYPTO_HASH_SHA256_HW_RT10xx_DCP`, 2656
 - `CRYPTO_HASH_SHA256_HW_STM32_HASH`, 2663
- MD5,
 - configuration,
 - compile-time, 93
 - memory footprint, 93
- PRESENT,
 - memory footprint, 1605
- RIPEMD-160,
 - configuration,
 - compile-time, 118
 - memory footprint, 118
- SEED,
 - configuration,
 - compile-time, 1294
 - memory footprint, 1294
- SHA-1,
 - configuration,
 - compile-time, 144
 - memory footprint, 144
- SHA-256,
 - configuration,
 - compile-time, 191
- memory footprint, 191
- SHA-512,
 - configuration,
 - compile-time, 235
 - memory footprint, 235
- SM3,
 - configuration,
 - compile-time, 381
 - memory footprint, 381
- Twofish,
 - configuration,
 - compile-time, 1553
 - memory footprint, 1553