CryptoAuthLib

v3.3.2

Chapter 1

CryptoAuthLib - Microchip CryptoAuthentication Library

Introduction

This library implements the APIs required to communicate with Microchip Security device. The family of devices supported currently are:

- ATSHA204A
- ATECC108A
- ATECC508A
- ATECC608A
- ATECC608B

The best place to start is with the Microchip Trust Platform

Online API documentation is at https://microchiptech.github.io/cryptoauthlib/

Latest software and examples can be found at:

- https://www.microchip.com/design-centers/security-ics/trust-platform
- http://www.microchip.com/SWLibraryWeb/product.aspx?product=CryptoAuthLib

Prerequisite hardware to run CryptoAuthLib examples:

- CryptoAuth Trust Platform Development Kit

Alternatively a Microchip MCU and Adapter Board:

- ATSAMR21 Xplained Pro or ATSAMD21 Xplained Pro
- CryptoAuth Xplained Pro Extension or CryptoAuthentication SOIC Socket Board to accept SOIC parts

For most development, using socketed top-boards is preferable until your configuration is well tested, then you can commit it to a CryptoAuth Xplained Pro Extension, for example. Keep in mind that once you lock a device, it will not be changeable.

Examples

- Watch CryptoAuthLib Documents for new examples coming online.
- Node Authentication Example Using Asymmetric PKI is a complete, all-in-one example demonstrating all the stages of crypto authentication starting from provisioning the Crypto Authentication device ATECC608/ATECC508A with keys and certificates to demonstrating an authentication sequence using asymmetric techniques. http://www.microchip.com/SWLibraryWeb/product.aspx?product=CryptoAuthLib

Configuration

In order to properly configured the library there must be a header file in your project named atca_config.h at minimum this needs to contain defines for the hal and device types being used. Most integrations have an configuration mechanism for generating this file. See the atca_config.h.in template which is configured by CMake for Linux, MacOS, & Windows projects.

An example of the configuration:

```
/* Cryptoauthlib Configuration File */
#ifndef ATCA_CONFIG_H
#define ATCA_CONFIG_H
/* Include HALS */
#define ATCA_HAL_I2C
/* Included device support */
#define ATCA_ATECC608_SUPPORT
/* \brief How long to wait after an initial wake failure for the POST to
 *         complete.
 * If Power-on self test (POST) is enabled, the self test will run on waking
 * from sleep or during power-on, which delays the wake reply.
 */
#ifndef ATCA_POST_DELAY_MSEC
#define ATCA_POST_DELAY_MSEC 25
#endif
#endif // ATCA_CONFIG_H
```

There are two major compiler defines that affect the operation of the library.

- ATCA_NO_POLL can be used to revert to a non-polling mechanism for device responses. Normally responses are polled for after sending a command, giving quicker response times. However, if ATCA_NO_POLL is defined, then the library will simply delay the max execution time of a command before reading the response.
- ATCA_NO_HEAP can be used to remove the use of malloc/free from the main library. This can be helpful for smaller MCUs that don't have a heap implemented. If just using the basic API, then there shouldn't be any code changes required. The lower-level API will no longer use the new/delete functions and the init/release functions should be used directly.

Release notes

See Release Notes

Host Device Support

CryptoAuthLib will run on a variety of platforms from small micro-controllers to desktop host systems. The current list of hardware abstraction layer support includes:

Rich OS Hosts:

- Linux Kit Protocol over HID USB
- Linux I2C
- Linux SPI
- Windows Kit Protocol over HID USB

Microcontrollers:

- Microchip AVR, SAM, & PIC families. See hal readme

If you have specific microcontrollers or Rich OS platforms you need support for, please contact us through the Microchip portal with your request.

CryptoAuthLib Architecture

Cryptoauthlib API documentation is at `https://microchiptech.github.io/cryptoauthlib/`

The library is structured to support portability to:

- multiple hardware/microcontroller platforms
- multiple environments including bare-metal, RTOS and Windows/Linux/MacOS
- multiple chip communication protocols (I2C, SPI, and SWI)

All platform dependencies are contained within the HAL (hardware abstraction layer).

Directory Structure

```
lib - primary library source code
lib/atcacert - certificate data and i/o methods
lib/calib - the Basic Cryptoauth API
lib/crypto - Software crypto implementations external crypto libraries support (primarily SHA1 and SHA256)
lib/hal - hardware abstraction layer code for supporting specific platforms
lib/host - support functions for common host-side calculations
lib/jwt - json web token functions
test - Integration test and examples. See test/cmd-processor.c for main() implementation.
For production code, test directories should be excluded by not compiling it
into a project, so it is up to the developer to include or not as needed.  Test
code adds significant bulk to an application - it's not intended to be included
in production code.
```

Tests

There is a set of integration tests found in the test directory which will at least partially demonstrate the use of the objects. Some tests may depend upon a certain device being configured in a certain way and may not work for all devices or specific configurations of the device.

The test/cmd-processor.c file contains a main() function for running the tests. It implements a command-line interface. Typing help will bring up the list of commands available.

One first selects a device type, with one of the following commands:

- 204 (ATSHA204A)
- 108 (ATECC108A)
- 508 (ATECC508A)
- 608 (ATECC608A/B)

From there the following unit test sweets are available:

- unit (test command builder functions)
- basic (test basic API functions)
- cio (test certification i/o functions)
- cd (test certificate data functions)
- util (test utility functions)
- crypto (test software crypto functions)

Tests available depend on the lock level of the device. The unit tests won't lock the config or data zones automatically to allow retesting at desired lock levels. Therefore, some commands will need to be repeated after locking to exercise all available tests.

Starting from a blank device, the sequence of commands to exercise all unit tests is:

```
unit
basic
lockcfg
unit
basic
lockdata
unit
basic
cio
cd
util
crypto
```

Using CryptoAuthLib (Microchip CryptoAuth Library)

The best place to start is with the `Microchip Trust Platform`

Also application examples are included as part of the Harmony 3 framework and can be copied from the Harmony Content Manager or found with the Harmony 3 Framework `Cryptoauthlib_apps`

Incorporating CryptoAuthLib in a Linux project using USB HID devices

The Linux HID HAL files use the Linux udev development software package.

To install the udev development package under Ubuntu Linux, please type the following command at the terminal window:

```
sudo apt-get install libudev-dev
```

This adds the udev development development software package to the Ubuntu Linux installation.

The Linux HID HAL files also require a udev rule to be added to change the permissions of the USB HID Devices. Please add a new udev rule for the Microchip CryptoAuth USB devices.

```
cd /etc/udev/rules.d
sudo touch mchp-cryptoauth.rules
```

Edit the mchp-cryptoauth.rules file and add the following line to the file:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2312", MODE="0666"
```

Chapter 2

License

mbedTLS Interface Functions that enable mbedtls objects to use cryptoauthlib functions

Replace mbedTLS ECDSA Functions with hardware acceleration & hardware key security.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

Replace mbedTLS ECDH Functions with hardware acceleration & hardware key security.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

Replace mbedTLS ECDSA Functions with hardware acceleration & hardware key security

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

mbedTLS Interface Functions that enable mbedtls objects to use cryptoauthlib functions

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

(c) 2018 Microchip Technology Inc. and its subsidiaries. You may use this software and any derivatives exclusively with Microchip products.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR ITS INTERACTION WITH MICROCHIP PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIPS TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

Chapter 3

openssl directory - Purpose

This directory contains the interfacing and wrapper functions to integrate openssl as the software crypto library.

Chapter 4

Application Support

This directory is for application specific implementation of various use cases.

Methods in this directory provide a simple API to perform potentially complex combinations of calls to the main library or API.

4.1 IP Protection with Symmetric Authentication

The IP protection can be easily integrated to the existing projects.The user project should include symmetric_authentication.c & symmetric_authentication.h files which contains the api

- **symmetric_authenticate()** - For Performing the authentication between host & device.

User Considerations

- The user should take care on how the master key should be stored on the MCU side.
- The api's in the file doesn't do the provisioning of the chip and user should take care of the provisioning.

With the provisioned cryptoauthentication device and after doing the cryptoauthlib initialisation,user should only be calling the function symmetric_authenticate() with its necessary parameters for the authentication. The returned authentication status should be used in the application.

Examples

For more information about IP protection and its example project refer `Microchip github`

4.2 PKCS11 Application Information

Setting up cryptoauthlib as a PKCS11 Provider for your system (LINUX)

These instructions are for building, installing and configuring cryptoauthlib as a pkcs11 provider. These instructions are for commonly available Linux systems with package managers.

Update libp11 on the system. The version should be at minimum 0.4.10

- Install the build dependendencies for the system:

  ```bash

  # Debian like systems

  $ sudo apt-get build-dep libengine-pkcs11-openssl1.1 ```

  ```bash

  # RPM based systems

  $ yum-builddep engine-pkcs11 ```

- Change to a sane directory

  ```bash cd ~ ```

- Get the latest version of libp11

  ```bash $ git clone https://github.com/OpenSC/libp11.git ```

- Rerun the build configuration tools:

  ``` $ cd libp11 $ ./bootstrap $ ./configure ```

- Build the library:

  ```bash $ make ```

- Install the library:

  ```bash $ sudo make install ```

Build and Install cryptoauthlib with PKCS11 support

- Install the build dependendencies for the system:

  ```bash

  # Debian like systems

  $ sudo apt-get install cmake libudev-dev ```

  ```bash
  ```

RPM based systems

$ yum install cmake $ yum install libudev-devel ```

- Change to a sane directory

  ```bash cd ~ ```

- Get the latest version of cryptoauthlib with PKCS11 support

  ```bash $ git clone –single-branch -b pkcs11 https://github.com/MicrochipTech/cryptoauthlib ```

- Rerun the build configuration tools:

  ```bash $ cd cryptoauthlib $ cmake . ```

- Build the library:

  ```bash $ make ```

- Install the library:

  ```bash $ sudo make install ```

Configuring the cryptoauthlib PKCS11 library

By default the following files will be created.

- /etc/cryptoauthlib/cryptoauthlib.conf

  ```text

## Cryptoauthlib Configuration File

filestore = /var/lib/cryptoauthlib ```

- /var/lib/cryptoauthlib/slot.conf.tmpl

  ```text

## Reserved Configuration for a device

## The objects in this file will be created and marked as undeletable

## These are processed in order. Configuration parameters must be comma

## delimited and may not contain spaces

interface = i2c,0xB0 freeslots = 1,2,3

## Slot 0 is the primary private key

object = private,device,0
  ```

Slot 10 is the certificate data for the device's public key

#object = certificate,device,10

Slot 12 is the intermedate/signer certificate data

#object = certificate,signer,12

Slot 15 is a public key

object = public,root,15 ```

cryptoauthlib.conf

This file provides the basic configuation information for the library. The only variable is "filestore" which is where cryptoauthlib will find device specific configuration and where it will store object files from pkcs11 operations.

slot.conf.tmpl

This is a template for device configuration files that cryptoauthlib will use to map devices and their resources into pkcs11 tokens and objects.

A device file must be named <pkcs11_slot_number>.conf

For a single device:
```
$ cd /var/lib/cryptoauthlib
$ cp slot.conf.tmpl 0.conf
```

Then edit 0.conf to match the device configuration being used.

interface Allows values: 'hid', 'i2c' If using i2c specify the address in hex for the device. This is in the device format (upper 7 bits define the address) so will not appear the same as the i2cdetect address (lower 7 bits)

freeslots This is a list of slots that may be used by the library when a pkcs11 operation that creates new objects is used. When the library is initialized it will scan for files of the form <pkcs11_slot_num>.<device_slot_num>.conf which defines the object using that device resource.

Using p11-kit-proxy

This is an optional step but is very helpful for using multiple pkcs11 libraries in a system. Detailed setup can be found at p11-glue
```
# Debian like systems
$ sudo apt-get install p11-kit
# RPM based systems
$ yum install p11-kit
```

- Create or edit the global configuration file `/etc/pkcs11/pkcs11.conf`. The directory `/etc/pkcs11` may require creation first.
  ```
  ```

This setting controls whether to load user configuration from the

∼/.config/pkcs11 directory. Possible values:

none: No user configuration

merge: Merge the user config over the system configuration (default)

only: Only user configuration, ignore system configuration

user-config: merge ```

- Create a module configuration file.
  - User module name (only available for a single user): `~/.config/pkcs11/modules/cryptoauthlib.module`
  - Global module name (available to the whole system): `/usr/share/p11-kit/modules/cryptoauthlib.modu`
    ``` module: /usr/lib/libcryptoauth.so critical: yes trust-policy: yes managed: yes log-calls: no ```

For more details on the configuration files see the configuration documentation.

Without using p11-kit-proxy

OpenSSL (via the libp11 project above) and p11tool support p11-kit-proxy natively so do not require additional set up if it is being used. If p11-kit-proxy is not being used then OpenSSL will have to be manually configured to use libp11 and cryptoauthlib

This requires editing the default openssl.cnf file. To locate the file being used by the system run the following command:

```
$ openssl version -a | grep OPENSSLDIR:
OPENSSLDIR: "/usr/lib/ssl"
```

This gives the default path where openssl is compiled to find the openssl.cnf file

In this case the file to edit will be /usr/lib/ssl/openssl.cnf

This line must be placed at the top, before any sections are defined:

```
openssl_conf = openssl_init
```

This should be added to the bottom of the file:

```
[openssl_init]
engines=engine_section
[engine_section]
pkcs11 = pkcs11_section
[pkcs11_section]
engine_id = pkcs11
# Wherever the engine installed by libp11 is. For example it could be:
# /usr/lib/arm-linux-gnueabihf/engines-1.1/libpkcs11.so
dynamic_path = /usr/lib/ssl/engines/libpkcs11.so
MODULE_PATH = /usr/lib/libcryptoauth.so
init = 0
```
```

Testing

To use p11tool it has to be installed:
```
# Debian like systems
$ sudo apt-get install gnutls-bin
# RPM based systems
$ yum install gnutls-utils
```

Note: If not using p11-kit-proxy then the provider has to be specified in p11tool calls:
```
$ p11tool --provider=/usr/lib/libcryptoauth.so
```

- Get the public key for a private key (as defined by the 0.conf file cited above):

  ```bash $ p11tool –export-pubkey "pkcs11:token=0123EE;object=device;type=private" warning: –login was not specified and it may be required for this operation. warning: no –outfile was specified and the public key will be printed on screen. -—–BEGIN PUBLIC KEY-—– MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQg↩ AE9wzUq1EUAoNrG01rXYjNd35mxKuA Ojw/kIlrNEBciSLLOTLjs/gvFS7N8AFXDK18vpxxu6ykzF2LRd7R↩ Y8yEFw== -—–END PUBLIC KEY-—– ```

- Get the public key and decode it using OpenSSL

  ```bash $ p11tool –export-pubkey "pkcs11:token=0123EE;object=device;type=private" | openssl pkey -pubin -text -noout warning: –login was not specified and it may be required for this operation. warning: no –outfile was specified and the public key will be printed on screen. Public-Key: (256 bit) pub: 04:f7:0c:d4:ab:51↩ :14:02:83:6b:1b:4d:6b:5d:88: cd:77:7e:66:c4:ab:80:3a:3c:3f:92:52:2b:34:40: 5c:89:22:cb:39:32:e3:b3:f8:2f↩ :15:2e:cd:f0:01: 57:0c:ad:7c:be:9c:71:bb:ac:a4:cc:5d:8b:45:de: d1:63:cc:84:17 ASN1 OID: prime256v1 NIST CURVE: P-256 ```

- Create a CSR for the private key

  ```bash $ openssl req -engine pkcs11 -key "pkcs11:token=0123EE;object=device;type=private" -keyform engine -new -out new_device.csr -subj "/CN=NEW CSR EXAMPLE" engine "pkcs11" set.

  $ cat new_device.csr -—–BEGIN CERTIFICATE REQUEST-—– MIHVMHwCAQAwGjEYMBYGA1UEAww↩ PTkVXIENTUiBFWEFNUExFMFkwEwYHKoZIzj0C AQYIKoZIzj0DAQcDQgAE9wzUq1EUAoNrG01rXYj↩ Nd35mxKuAOjw/kIlrNEBciSLL OTLjs/gvFS7N8AFXDK18vpxxu6ykzF2LRd7RY8yEF6AAMAoGCCqGS↩ M49BAMCA0kA MEYCIQDUPeLfPcOwtZxYJDYXPdl2UhpReVn6kK2lKCCX6byM8QIhAIfqfnggtcCi W21x↩ LAzabr8A4mHyfIIQ1ofYBg8QO9jZ -—–END CERTIFICATE REQUEST-—– ```

- Verify the newly created csr

  ```bash $ openssl req -in new_device.csr -verify -text -noout verify OK Certificate Request: Data: Version: 1 (0x0) Subject: CN = NEW CSR EXAMPLE Subject Public Key Info: Public Key Algorithm: id-ecPublicKey Public-Key: (256 bit) pub: 04:f7:0c:d4:ab:51:14:02:83:6b:1b:4d:6b:5d:88: cd:77:7e:66:c4:ab:80:3a:3c:3f↩ :92:52:2b:34:40: 5c:89:22:cb:39:32:e3:b3:f8:2f:15:2e:cd:f0:01: 57:0c:ad:7c:be:9c:71:bb:ac:a4:cc:5d:8b:45↩ :de: d1:63:cc:84:17 ASN1 OID: prime256v1 NIST CURVE: P-256 Attributes: a0:00 Signature Algorithm↩ : ecdsa-with-SHA256 30:46:02:21:00:d4:3d:e2:df:3d:c3:b0:b5:9c:58:24:36:17: 3d:d9:76:52:1a:51:79:59:fa↩ :90:ad:a5:28:20:97:e9:bc:8c: f1:02:21:00:87:ea:7e:78:20:b5:c0:a2:5b:6d:71:2c:0c:da: 6e:bf:00:e2:61:f2:7c↩ :82:10:d6:87:d8:06:0f:10:3b:d8:d9 ```

## 4.3 Secure boot using ATECC608

The SecureBoot command is a new feature on the ATECC608A device compared to earlier CryptoAuthentication devices from Microchip. This feature helps the MCU to identify fraudulent code installed on it. When this feature is implemented, the MCU can send a firmware digest and signature to the ATECC608. The ATECC608 validates this information (ECDSA verify) and responds to host with a yes or no answer.

The ATECC608 provides options to reduce the firmware verification time by storing the signature or digest after a good full verification (FullStore mode of the SecureBoot command).

- When the ATECC608 stores the digest (SecureBootMode is FullDig), the host only needs to send the firmware digest, which is compared to the stored copy. This skips the comparatively lengthy ECDSA verify, speeding up the secure boot process.
- When the ATECC608 stores the signature (SecureBootMode is FullSig), the host only needs to send the firmware digest, which is verified against the stored signature using ECDSA. This saves time by not needing to send the signature in the command over the bus.

The ATECC608 also provides wire protection features for the SecureBoot command, which can be used to encrypt the digest being sent from the host to the ATECC608 and add a MAC to the verify result coming back to the host so it can't be forced to a success state. This feature makes use of a shared secret between the host and ATECC608, called the IO protection key.

The secure boot feature can be easily integrated to an existing project. The project should include the following files from the secure_boot folder:

- secure_boot.c
- secure_boot.h
- secure_boot_memory.h
- io_protection_key.h

The project should also implement the following platform-specific APIs:

- secure_boot_init_memory()
- secure_boot_read_memory()
- secure_boot_deinit_memory()
- secure_boot_mark_full_copy_completion()
- secure_boot_check_full_copy_completion()
- io_protection_get_key()
- io_protection_set_key()

The project can set the secure boot configuration with the following defines:

- SECURE_BOOT_CONFIGURATION
- SECURE_BOOT_DIGEST_ENCRYPT_ENABLED
- SECURE_BOOT_UPGRADE_SUPPORT

The secure boot process is performed by initializing CryptoAuthLib and calling the secure_boot_process() function.

## Implementation Considerations

- Need to perform SHA256 calculations on the host. CryptoAuthLib provides a software implementation in lib/crypto/atca_crypto_sw_sha2.c
- When using the wire protection features:
  - The host needs to be able to generate a nonce (number used once). This is the NumIn parameter to the Nonce command that is sent before the SecureBoot command. The ATECC608 can not be used to generate NumIn, but it should come from a good random or non-repeating source in the host.
  - If the host has any protected internal memory, it should be used to store its copy of the IO protection key.
- Secure boot depends on proper protections of the boot loader code in the host. If the code can be easily changed, then the secure boot process can be easily skipped. Boot loader should ideally be stored in an immutable (unchangeable) location like a boot ROM or write-protected flash.
- Note that these APIs don't provision the ATECC608. They assume the ATECC608 has already been configured and provisioned with the necessary keys for secure boot.

## Examples

For more information about secure boot, please see the example implementation project and documentation at: https://github.com/MicrochipTech/cryptoauth_usecase_secureboot

# Chapter 5

# Module Index

## 5.1 Modules

Here is a list of all modules:

# Chapter 6

# Data Structure Index

## 6.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 7

# File Index

## 7.1 File List

Here is a list of all files with brief descriptions:

# Chapter 8

# Module Documentation

## 8.1 Basic Crypto API methods (atcab_)

These methods provide the most convenient, simple API to CryptoAuth chips.

### Macros

- #define atcab_get_addr(...) calib_get_addr(__VA_ARGS__)
- #define atca_execute_command(...) calib_execute_command(__VA_ARGS__)
- #define SHA_CONTEXT_MAX_SIZE (109)

### Functions

- ATCA_STATUS atcab_version (char ∗ver_str)

    *basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.*
- ATCA_STATUS atcab_init_ext (ATCADevice ∗device, ATCAIfaceCfg ∗cfg)

    *Creates and initializes a ATCADevice context.*
- ATCA_STATUS atcab_init (ATCAIfaceCfg ∗cfg)

    *Creates a global ATCADevice object used by Basic API.*
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

    *Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.*
- ATCA_STATUS atcab_release_ext (ATCADevice ∗device)

    *release (free) the an ATCADevice instance.*
- ATCA_STATUS atcab_release (void)

    *release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.*
- ATCADevice atcab_get_device (void)

    *Get the global device object.*
- ATCADeviceType atcab_get_device_type_ext (ATCADevice device)

    *Get the selected device type of rthe device context.*
- ATCADeviceType atcab_get_device_type (void)

    *Get the current device type configured for the global ATCADevice.*
- uint8_t atcab_get_device_address (ATCADevice device)

    *Get the current device address based on the configured device and interface.*

- bool atcab_is_ca_device (ATCADeviceType dev_type)

  *Check whether the device is cryptoauth device.*
- bool atcab_is_ta_device (ATCADeviceType dev_type)

  *Check whether the device is Trust Anchor device.*
- ATCA_STATUS atcab_aes_cbc_init_ext (ATCADevice device, atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  *Initialize context for AES CBC operation.*
- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  *Initialize context for AES CBC operation.*
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  *Encrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  *Decrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_cbcmac_init_ext (ATCADevice device, atca_aes_cbcmac_ctx_t ∗ctx, uint16_↩ t key_id, uint8_t key_block)

  *Initialize context for AES CBC-MAC operation.*
- ATCA_STATUS atcab_aes_cbcmac_init (atca_aes_cbcmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  *Initialize context for AES CBC-MAC operation.*
- ATCA_STATUS atcab_aes_cbcmac_update (atca_aes_cbcmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_↩ t data_size)

  *Calculate AES CBC-MAC with key stored within ECC608A device. calib_aes_cbcmac_init() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_cbcmac_finish (atca_aes_cbcmac_ctx_t ∗ctx, uint8_t ∗mac, uint32_t mac_size)

  *Finish a CBC-MAC operation returning the CBC-MAC value. If the data provided to the calib_aes_cbcmac_update() function has incomplete block this function will return an error code.*
- ATCA_STATUS atcab_aes_cmac_init_ext (ATCADevice device, atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  *Initialize a CMAC calculation using an AES-128 key in the device.*
- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  *Initialize a CMAC calculation using an AES-128 key in the device.*
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_↩ size)

  *Add data to an initialized CMAC calculation.*
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t ∗ctx, uint8_t ∗cmac, uint32_t cmac_size)

  *Finish a CMAC operation returning the CMAC value.*
- ATCA_STATUS atcab_aes_ctr_init_ext (ATCADevice device, atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t ∗iv)

  *Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.*
- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_↩ t counter_size, const uint8_t ∗iv)

  *Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.*
- ATCA_STATUS atcab_aes_ctr_init_rand_ext (ATCADevice device, atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  *Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.*
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  *Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.*

- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *input, uint8_t *output)

  *Process a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *plaintext, uint8_t *ciphertext)

  *Encrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *ciphertext, uint8_t *plaintext)

  *Decrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t *ctx)

  *Increments AES CTR counter value.*
- ATCA_STATUS atcab_aes_ccm_init_ext (ATCADevice device, atca_aes_ccm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t *iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  *Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.*
- ATCA_STATUS atcab_aes_ccm_init (atca_aes_ccm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t *iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  *Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.*
- ATCA_STATUS atcab_aes_ccm_init_rand_ext (ATCADevice device, atca_aes_ccm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t *iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  *Initialize context for AES CCM operation with a random nonce.*
- ATCA_STATUS atcab_aes_ccm_init_rand (atca_aes_ccm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t *iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  *Initialize context for AES CCM operation with a random nonce.*
- ATCA_STATUS atcab_aes_ccm_aad_update (atca_aes_ccm_ctx_t *ctx, const uint8_t *aad, size_t aad_size)

  *Process Additional Authenticated Data (AAD) using CCM mode and a key within the ATECC608A device.*
- ATCA_STATUS atcab_aes_ccm_aad_finish (atca_aes_ccm_ctx_t *ctx)

  *Finish processing Additional Authenticated Data (AAD) using CCM mode.*
- ATCA_STATUS atcab_aes_ccm_encrypt_update (atca_aes_ccm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

  *Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_ccm_decrypt_update (atca_aes_ccm_ctx_t *ctx, const uint8_t *ciphertext, uint32_t ciphertext_size, uint8_t *plaintext)

  *Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_ccm_encrypt_finish (atca_aes_ccm_ctx_t *ctx, uint8_t *tag, uint8_t *tag_size)

  *Complete a CCM encrypt operation returning the authentication tag.*
- ATCA_STATUS atcab_aes_ccm_decrypt_finish (atca_aes_ccm_ctx_t *ctx, const uint8_t *tag, bool *is_verified)

  *Complete a CCM decrypt operation authenticating provided tag.*
- ATCA_STATUS atcab_pbkdf2_sha256_ext (ATCADevice device, const uint32_t iter, const uint16_t slot, const uint8_t *salt, const size_t salt_len, uint8_t *result, size_t result_len)

  *Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.*
- ATCA_STATUS atcab_pbkdf2_sha256 (const uint32_t iter, const uint16_t slot, const uint8_t *salt, const size_t salt_len, uint8_t *result, size_t result_len)

  *Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.*
- ATCA_STATUS _atcab_exit (void)
- ATCA_STATUS atcab_wakeup (void)

  *wakeup the CryptoAuth device*

- ATCA_STATUS atcab_idle (void)

  *idle the CryptoAuth device*
- ATCA_STATUS atcab_sleep (void)

  *invoke sleep on the CryptoAuth device*
- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t *size)

  *Gets the size of the specified zone in bytes.*
- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t *aes_in, uint8_t *aes_out)

  *Compute the AES-128 encrypt, decrypt, or GFM calculation.*
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_t *ciphertext)

  *Perform an AES-128 encrypt operation with a key in the device.*
- ATCA_STATUS atcab_aes_encrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_t *ciphertext)

  *Perform an AES-128 encrypt operation with a key in the device.*
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

  *Perform an AES-128 decrypt operation with a key in the device.*
- ATCA_STATUS atcab_aes_decrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

  *Perform an AES-128 decrypt operation with a key in the device.*
- ATCA_STATUS atcab_aes_gfm (const uint8_t *h, const uint8_t *input, uint8_t *output)

  *Perform a Galois Field Multiply (GFM) operation.*
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, const uint8_t *iv, size_t iv_size)

  *Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.*
- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t *free_field, size_t free_field_size, uint8_t *iv)

  *Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.*
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *aad, uint32_t aad_size)

  *Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.*
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

  *Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t *ctx, uint8_t *tag, size_t tag_size)

  *Complete a GCM encrypt operation returning the authentication tag.*
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *ciphertext, uint32_t ciphertext_size, uint8_t *plaintext)

  *Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t *ctx, const uint8_t *tag, size_t tag_size, bool *is_verified)

  *Complete a GCM decrypt operation verifying the authentication tag.*
- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t *challenge, const uint8_t *response, const uint8_t *other_data)

  *Compares a MAC response with input values.*
- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t *counter_value)

  *Compute the Counter functions.*
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t *counter_value)

  *Increments one of the device's monotonic counters.*
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t *counter_value)

*Read one of the device's monotonic counters.*
- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t key_id, const uint8_t ∗mac)

  *Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.*
- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)

  *Base function for generating premaster secret key using ECDH.*
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)

  *ECDH command with a private key in a slot and the premaster secret is returned in the clear.*
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id, const uint8_t num_in[(20)])

  *ECDH command with a private key in a slot and the premaster secret is read from the next slot.*
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  *ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.*
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t ∗public_key, uint8_t ∗pms)

  *ECDH command with a private key in TempKey and the premaster secret is returned in the clear.*
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  *ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.*
- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t ∗other_data, uint8_t other_data↩ _size)

  *Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.*
- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t ∗other_data, uint8_↩ t ∗public_key)

  *Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.*
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t ∗public_key)

  *Issues GenKey command, which generates a new random private key in slot/handle and returns the public key.*
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t ∗public_key)

  *Uses GenKey command to calculate the public key from an existing private key in a slot.*
- ATCA_STATUS atcab_get_pubkey_ext (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  *Uses GenKey command to calculate the public key from an existing private key in a slot.*
- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t ∗digest)

  *Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.*
- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  *Issues an Info command, which return internal device information and can control GPIO and the persistent latch.*
- ATCA_STATUS atcab_info (uint8_t ∗revision)

  *Use the Info command to get the device revision (DevRev).*
- ATCA_STATUS atcab_info_set_latch (bool state)

  *Use the Info command to set the persistent latch state for an ATECC608 device.*
- ATCA_STATUS atcab_info_get_latch (bool ∗state)

  *Use the Info command to get the persistent latch current state for an ATECC608 device.*
- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

  *Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.*
- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

  *The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.*
- ATCA_STATUS atcab_lock_config_zone (void)

*Unconditionally (no CRC required) lock the config zone.*

- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

  *Lock the config zone with summary CRC.*
- ATCA_STATUS atcab_lock_data_zone (void)

  *Unconditionally (no CRC required) lock the data zone (slots and OTP). for CryptoAuth devices and lock the setup for Trust Anchor device.*
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

  *Lock the data zone (slots and OTP) with summary CRC.*
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

  *Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1) (for cryptoauth devices) or Lock an individual handle in shared data element on an Trust Anchor device (for Trust Anchor devices).*
- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  *Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.*
- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  *Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.*
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)

  *Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.*
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)

  *Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.*
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)

  *Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.*
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)

  *Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.*
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)

  *Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.*
- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[(20)])

  *Executes PrivWrite command, to write externally generated ECC private keys into the device.*
- ATCA_STATUS atcab_random (uint8_t ∗rand_out)

  *Executes Random command, which generates a 32 byte random number from the device.*
- ATCA_STATUS atcab_random_ext (ATCADevice device, uint8_t ∗rand_out)

  *Executes Random command, which generates a 32 byte random number from the device.*
- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

  *Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.*
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

  *Executes Read command, which reads the configuration zone to see if the specified zone is locked.*
- ATCA_STATUS atcab_is_config_locked (bool ∗is_locked)

  *This function check whether configuration zone is locked or not.*
- ATCA_STATUS atcab_is_data_locked (bool ∗is_locked)

  *This function check whether data/setup zone is locked or not.*
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

  *This function check whether slot/handle is locked or not.*
- ATCA_STATUS atcab_is_private_ext (ATCADevice device, uint16_t slot, bool ∗is_private)

  *Check to see if the key is a private key or not.*
- ATCA_STATUS atcab_is_private (uint16_t slot, bool ∗is_private)

- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t *data, size_t length)

  *Used to read an arbitrary number of bytes from any zone configured for clear reads.*
- ATCA_STATUS atcab_read_serial_number (uint8_t *serial_number)

  *This function returns serial number of the device.*
- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t *public_key)

  *Executes Read command to read an ECC P256 public key from a slot configured for clear reads.*
- ATCA_STATUS atcab_read_pubkey_ext (ATCADevice device, uint16_t slot, uint8_t *public_key)

  *Executes Read command to read an ECC P256 public key from a slot configured for clear reads.*
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t *sig)

  *Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.*
- ATCA_STATUS atcab_read_config_zone (uint8_t *config_data)

  *Executes Read command to read the complete device configuration zone.*
- ATCA_STATUS atcab_cmp_config_zone (uint8_t *config_data, bool *same_config)

  *Compares a specified configuration zone with the configuration zone currently on the device.*
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  *Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.*
- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

  *Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.*
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  *Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.*
- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t *result)

  *Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608 chip.*
- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t *data_in, uint8_t *data_out, uint16_t *data_out_size)

  *Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.*
- ATCA_STATUS atcab_sha_start (void)

  *Executes SHA command to initialize SHA-256 calculation engine.*
- ATCA_STATUS atcab_sha_update (const uint8_t *message)

  *Executes SHA command to add 64 bytes of message data to the current context.*
- ATCA_STATUS atcab_sha_end (uint8_t *digest, uint16_t length, const uint8_t *message)

  *Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sha_read_context (uint8_t *context, uint16_t *context_size)

  *Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.*
- ATCA_STATUS atcab_sha_write_context (const uint8_t *context, uint16_t context_size)

  *Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.*
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t *message, uint8_t *digest)

  *Use the SHA command to compute a SHA-256 digest.*
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t *data, size_t data_size, uint8_t *digest)

  *Use the SHA command to compute a SHA-256 digest.*
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t *ctx)

  *Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.*
- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

*Add message data to a SHA context for performing a hardware SHA-256 operation on a device.*
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t *ctx, uint8_t *digest)

  *Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.*
- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t *ctx, uint16_t key_slot)

  *Executes SHA command to start an HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t *ctx, const uint8_t *data, size_t data↩_size)

  *Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t *ctx, uint8_t *digest, uint8_t target)

  *Executes SHA command to complete a HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sha_hmac (const uint8_t *data, size_t data_size, uint16_t key_slot, uint8_t *digest, uint8_t target)

  *Use the SHA command to compute an HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sha_hmac_ext (ATCADevice device, const uint8_t *data, size_t data_size, uint16_t key_slot, uint8_t *digest, uint8_t target)

  *Use the SHA command to compute an HMAC/SHA-256 operation.*
- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t *signature)

  *Executes the Sign command, which generates a signature using the ECDSA algorithm.*
- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  *Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.*
- ATCA_STATUS atcab_sign_ext (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_↩t *signature)

  *Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.*
- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t *signature)

  *Executes Sign command to sign an internally generated message.*
- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  *Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).*
- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t *signature, const uint8_t *public↩_key, const uint8_t *other_data, uint8_t *mac)

  *Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.*
- ATCA_STATUS atcab_verify_extern (const uint8_t *message, const uint8_t *signature, const uint8_↩t *public_key, bool *is_verified)

  *Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.*
- ATCA_STATUS atcab_verify_extern_ext (ATCADevice device, const uint8_t *message, const uint8_↩t *signature, const uint8_t *public_key, bool *is_verified)

  *Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.*
- ATCA_STATUS atcab_verify_extern_mac (const uint8_t *message, const uint8_t *signature, const uint8_t *public_key, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  *Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.*
- ATCA_STATUS atcab_verify_stored (const uint8_t *message, const uint8_t *signature, uint16_t key_id, bool *is_verified)

  *Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩Key for other devices.*

- ATCA_STATUS atcab_verify_stored_ext (ATCADevice device, const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  *Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.*
- ATCA_STATUS atcab_verify_stored_mac (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  *Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.*
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  *Executes the Verify command in Validate mode to validate a public key stored in a slot.*
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  *Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.*
- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  *Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.*
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)

  *Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.*
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

  *Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).*
- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t ∗public_key)

  *Uses the write command to write a public key to a slot in the proper format.*
- ATCA_STATUS atcab_write_config_zone (const uint8_t ∗config_data)

  *Executes the Write command, which writes the configuration zone.*
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  *Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.*
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

  *Initialize one of the monotonic counters in device with a specific value.*

## Variables

- ATCADevice _gDevice

- ATCA_STATUS atcab_printbin (uint8_t ∗binary, size_t bin_len, bool add_space)

- const char ∗ atca_basic_aes_gcm_version
- ATCA_STATUS calib_aes_gcm_init (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv, size_t iv_size)

  *Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.*
- ATCA_STATUS calib_aes_gcm_init_rand (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t ∗free_field, size_t free_field_size, uint8_t ∗iv)

  *Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.*

- ATCA_STATUS calib_aes_gcm_aad_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗aad, uint32_t aad_size)

  *Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.*
- ATCA_STATUS calib_aes_gcm_encrypt_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

  *Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS calib_aes_gcm_encrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint8_t ∗tag, size_t tag_size)

  *Complete a GCM encrypt operation returning the authentication tag.*
- ATCA_STATUS calib_aes_gcm_decrypt_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  *Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.*
- ATCA_STATUS calib_aes_gcm_decrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗tag, size_t tag_size, bool ∗is_verified)

  *Complete a GCM decrypt operation verifying the authentication tag.*
- #define ATCA_AES_GCM_IV_STD_LENGTH 12
- typedef struct atca_aes_gcm_ctx atca_aes_gcm_ctx_t

### 8.1.1 Detailed Description

These methods provide the most convenient, simple API to CryptoAuth chips.

### 8.1.2 Macro Definition Documentation

#### 8.1.2.1 ATCA_AES_GCM_IV_STD_LENGTH

```
#define ATCA_AES_GCM_IV_STD_LENGTH 12
```

8.1.2.2 atca_execute_command

```
#define atca_execute_command(
              ... ) calib_execute_command(__VA_ARGS__)
```

8.1.2.3 atcab_get_addr

```
#define atcab_get_addr(
              ... ) calib_get_addr(__VA_ARGS__)
```

8.1.2.4 SHA_CONTEXT_MAX_SIZE

```
#define SHA_CONTEXT_MAX_SIZE (109)
```

8.1.3 Typedef Documentation

8.1.3.1 atca_aes_gcm_ctx_t

```
typedef struct atca_aes_gcm_ctx atca_aes_gcm_ctx_t
```

Context structure for AES GCM operations.

8.1.4 Function Documentation

8.1.4.1 _atcab_exit()

```
ATCA_STATUS _atcab_exit (
            void )
```

8.1.4.2 atcab_aes()

```
ATCA_STATUS atcab_aes (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * aes_in,
            uint8_t * aes_out )
```

Compute the AES-128 encrypt, decrypt, or GFM calculation.

Parameters

in	*mode*	The mode for the AES command.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*aes_in*	Input data to the AES command (16 bytes).
out	*aes_out*	Output data from the AES command is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.3 atcab_aes_cbc_decrypt_block()

```
ATCA_STATUS atcab_aes_cbc_decrypt_block (
            atca_aes_cbc_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Decrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.

Parameters

in	*ctx*	AES CBC context.
in	*ciphertext*	Ciphertext to be decrypted (16 bytes).
out	*plaintext*	Decrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.4 atcab_aes_cbc_encrypt_block()

```
ATCA_STATUS atcab_aes_cbc_encrypt_block (
            atca_aes_cbc_ctx_t * ctx,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Encrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.

Parameters

in	*ctx*	AES CBC context.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
out	*ciphertext*	Encrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.5 atcab_aes_cbc_init()

```
ATCA_STATUS atcab_aes_cbc_init (
            atca_aes_cbc_ctx_t * ctx,
            uint16_t key_id,
```

```
uint8_t key_block,
const uint8_t * iv )
```

Initialize context for AES CBC operation.

Parameters

in	*ctx*	AES CBC context to be initialized
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.6 atcab_aes_cbc_init_ext()

```
ATCA_STATUS atcab_aes_cbc_init_ext (
            ATCADevice device,
            atca_aes_cbc_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv )
```

Initialize context for AES CBC operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES CBC context to be initialized
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.7 atcab_aes_cbcmac_finish()

```
ATCA_STATUS atcab_aes_cbcmac_finish (
            atca_aes_cbcmac_ctx_t * ctx,
            uint8_t * mac,
            uint32_t mac_size )
```

Finish a CBC-MAC operation returning the CBC-MAC value. If the data provided to the calib_aes_cbcmac_update() function has incomplete block this function will return an error code.

Parameters

in	*ctx*	AES-128 CBC-MAC context.
out	*mac*	CBC-MAC is returned here.
in	*mac_size*	Size of CBC-MAC requested in bytes (max 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.8 atcab_aes_cbcmac_init()

```
ATCA_STATUS atcab_aes_cbcmac_init (
            atca_aes_cbcmac_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block )
```

Initialize context for AES CBC-MAC operation.

Parameters

in	*ctx*	AES CBC-MAC context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.9 atcab_aes_cbcmac_init_ext()

```
ATCA_STATUS atcab_aes_cbcmac_init_ext (
            ATCADevice device,
            atca_aes_cbcmac_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block )
```

Initialize context for AES CBC-MAC operation.

Parameters

in	*ctx*	AES CBC-MAC context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.10 atcab_aes_cbcmac_update()

```
ATCA_STATUS atcab_aes_cbcmac_update (
            atca_aes_cbcmac_ctx_t * ctx,
            const uint8_t * data,
            uint32_t data_size )
```

Calculate AES CBC-MAC with key stored within ECC608A device. calib_aes_cbcmac_init() should be called before the first use of this function.

Parameters

in	*ctx*	AES CBC-MAC context structure.
in	*data*	Data to be added for AES CBC-MAC calculation.
in	*data_size*	Data length in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.11 atcab_aes_ccm_aad_finish()

```
ATCA_STATUS atcab_aes_ccm_aad_finish (
            atca_aes_ccm_ctx_t * ctx )
```

Finish processing Additional Authenticated Data (AAD) using CCM mode.

This function is called once all additional authentication data has been added into ccm calculation through calib_↩ aes_ccm_aad_update() function.

This is an internal function, this function is called by the calib_aes_ccm_update()

Parameters

in	*ctx*	AES CCM context

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.12 atcab_aes_ccm_aad_update()

```
ATCA_STATUS atcab_aes_ccm_aad_update (
            atca_aes_ccm_ctx_t * ctx,
            const uint8_t * aad,
            size_t aad_size )
```

Process Additional Authenticated Data (AAD) using CCM mode and a key within the ATECC608A device.

This can be called multiple times. calib_aes_ccm_init() or calib_aes_ccm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before calib_aes_ccm_encrypt_↩ update() or calib_aes_ccm_decrypt_update().

Parameters

in	*ctx*	AES CCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.13 atcab_aes_ccm_decrypt_finish()

```
ATCA_STATUS atcab_aes_ccm_decrypt_finish (
            atca_aes_ccm_ctx_t * ctx,
            const uint8_t * tag,
            bool * is_verified )
```

Complete a CCM decrypt operation authenticating provided tag.

Parameters

in	*ctx*	AES CCM context structure.
in	*tag*	Tag to be authenticated.
out	*is_verified*	Value is set to true if the tag is authenticated else the value is set to false.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.14 atcab_aes_ccm_decrypt_update()

```
ATCA_STATUS atcab_aes_ccm_decrypt_update (
            atca_aes_ccm_ctx_t * ctx,
```

```
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm↵_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CCM context structure.
in	*ciphertext*	Data to be processed.
out	*plaintext*	Output data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.15 atcab_aes_ccm_encrypt_finish()

```
ATCA_STATUS atcab_aes_ccm_encrypt_finish (
            atca_aes_ccm_ctx_t * ctx,
            uint8_t * tag,
            uint8_t * tag_size )
```

Complete a CCM encrypt operation returning the authentication tag.

Parameters

in	*ctx*	AES CCM context structure.
out	*tag*	Authentication tag is returned here.
out	*tag_size*	Tag size in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.16 atcab_aes_ccm_encrypt_update()

```
ATCA_STATUS atcab_aes_ccm_encrypt_update (
            atca_aes_ccm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm↵_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CCM context structure.
in	*plaintext*	Data to be processed.
out	*ciphertext*	Output data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.17 atcab_aes_ccm_init()

```
ATCA_STATUS atcab_aes_ccm_init (
            atca_aes_ccm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t * iv,
            size_t iv_size,
            size_t aad_size,
            size_t text_size,
            size_t tag_size )
```

Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*ctx*	AES CCM context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Nonce to be fed into the AES CCM calculation.
in	*iv_size*	Size of iv.
in	*aad_size*	Size of Additional authtication data.
in	*text_size*	Size of plaintext/ciphertext to be processed.
in	*tag_size*	Prefered size of tag.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.18 atcab_aes_ccm_init_ext()

```
ATCA_STATUS atcab_aes_ccm_init_ext (
            ATCADevice device,
            atca_aes_ccm_ctx_t * ctx,
            uint16_t key_id,
```

```
uint8_t key_block,
uint8_t * iv,
size_t iv_size,
size_t aad_size,
size_t text_size,
size_t tag_size )
```

Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*ctx*	AES CCM context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Nonce to be fed into the AES CCM calculation.
in	*iv_size*	Size of iv.
in	*aad_size*	Size of Additional authtication data.
in	*text_size*	Size of plaintext/ciphertext to be processed.
in	*tag_size*	Prefered size of tag.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.19 atcab_aes_ccm_init_rand()

```
ATCA_STATUS atcab_aes_ccm_init_rand (
            atca_aes_ccm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t * iv,
            size_t iv_size,
            size_t aad_size,
            size_t text_size,
            size_t tag_size )
```

Initialize context for AES CCM operation with a random nonce.

Parameters

in	*ctx*	AES CCM context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
out	*iv*	Nonce used for AES CCM calculation is returned here.
in	*iv_size*	Size of iv.
in	*aad_size*	Size of Additional authtication data.
in	*text_size*	Size of plaintext/ciphertext to be processed.
in	*tag_size*	Prefered size of tag.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.20 atcab_aes_ccm_init_rand_ext()

```
ATCA_STATUS atcab_aes_ccm_init_rand_ext (
            ATCADevice device,
            atca_aes_ccm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t * iv,
            size_t iv_size,
            size_t aad_size,
            size_t text_size,
            size_t tag_size )
```

Initialize context for AES CCM operation with a random nonce.

Parameters

in	*ctx*	AES CCM context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
out	*iv*	Nonce used for AES CCM calculation is returned here.
in	*iv_size*	Size of iv.
in	*aad_size*	Size of Additional authtication data.
in	*text_size*	Size of plaintext/ciphertext to be processed.
in	*tag_size*	Prefered size of tag.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.21 atcab_aes_cmac_finish()

```
ATCA_STATUS atcab_aes_cmac_finish (
            atca_aes_cmac_ctx_t * ctx,
            uint8_t * cmac,
            uint32_t cmac_size )
```

Finish a CMAC operation returning the CMAC value.

Parameters

in	*ctx*	AES-128 CMAC context.
out	*cmac*	CMAC is returned here.
in	*cmac_size*	Size of CMAC requested in bytes (max 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.22 atcab_aes_cmac_init()

```
ATCA_STATUS atcab_aes_cmac_init (
            atca_aes_cmac_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block )
```

Initialize a CMAC calculation using an AES-128 key in the device.

Parameters

in	*ctx*	AES-128 CMAC context.
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.23 atcab_aes_cmac_init_ext()

```
ATCA_STATUS atcab_aes_cmac_init_ext (
            ATCADevice device,
            atca_aes_cmac_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block )
```

Initialize a CMAC calculation using an AES-128 key in the device.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES-128 CMAC context.
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.24 atcab_aes_cmac_update()

```
ATCA_STATUS atcab_aes_cmac_update (
            atca_aes_cmac_ctx_t * ctx,
            const uint8_t * data,
            uint32_t data_size )
```

Add data to an initialized CMAC calculation.

Parameters

in	*ctx*	AES-128 CMAC context.
in	*data*	Data to be added.
in	*data_size*	Size of the data to be added in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.25 atcab_aes_ctr_block()

```
ATCA_STATUS atcab_aes_ctr_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * input,
            uint8_t * output )
```

Process a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*input*	Input data to be processed (16 bytes).
out	*output*	Output data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

8.1.4.26 atcab_aes_ctr_decrypt_block()

```
ATCA_STATUS atcab_aes_ctr_decrypt_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Decrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*ciphertext*	Ciphertext to be decrypted (16 bytes).
out	*plaintext*	Decrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

8.1.4.27 atcab_aes_ctr_encrypt_block()

```
ATCA_STATUS atcab_aes_ctr_encrypt_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Encrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
out	*ciphertext*	Encrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

8.1.4.28 atcab_aes_ctr_increment()

```
ATCA_STATUS atcab_aes_ctr_increment (
            atca_aes_ctr_ctx_t * ctx )
```

Increments AES CTR counter value.

Parameters

in,out	*ctx*	AES CTR context

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.29 atcab_aes_ctr_init()

```
ATCA_STATUS atcab_aes_ctr_init (
            atca_aes_ctr_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            const uint8_t * iv )
```

Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
in	*iv*	Initialization vector (concatenation of nonce and counter) 16 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.30 atcab_aes_ctr_init_ext()

```
ATCA_STATUS atcab_aes_ctr_init_ext (
            ATCADevice device,
            atca_aes_ctr_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            const uint8_t * iv )
```

Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot/handles or in TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
in	*iv*	Initialization vector (concatenation of nonce and counter) 16 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.31 atcab_aes_ctr_init_rand()

```
ATCA_STATUS atcab_aes_ctr_init_rand (
            atca_aes_ctr_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            uint8_t * iv )
```

Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
out	*iv*	Initialization vector (concatenation of nonce and counter) is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.32 atcab_aes_ctr_init_rand_ext()

```
ATCA_STATUS atcab_aes_ctr_init_rand_ext (
            ATCADevice device,
            atca_aes_ctr_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            uint8_t * iv )
```

Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
out	*iv*	Initialization vector (concatenation of nonce and counter) is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.33 atcab_aes_decrypt()

```
ATCA_STATUS atcab_aes_decrypt (
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Perform an AES-128 decrypt operation with a key in the device.

Parameters

in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*ciphertext*	Input ciphertext to be decrypted (16 bytes).
out	*plaintext*	Output plaintext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.34 atcab_aes_decrypt_ext()

```
ATCA_STATUS atcab_aes_decrypt_ext (
            ATCADevice device,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Perform an AES-128 decrypt operation with a key in the device.

Parameters

in	*device*	Device context pointer
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*ciphertext*	Input ciphertext to be decrypted (16 bytes).
out	*plaintext*	Output plaintext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.35 atcab_aes_encrypt()

```
ATCA_STATUS atcab_aes_encrypt (
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Perform an AES-128 encrypt operation with a key in the device.

Parameters

in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*plaintext*	Input plaintext to be encrypted (16 bytes).
out	*ciphertext*	Output ciphertext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.36 atcab_aes_encrypt_ext()

```
ATCA_STATUS atcab_aes_encrypt_ext (
            ATCADevice device,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Perform an AES-128 encrypt operation with a key in the device.

Parameters

in	*device*	Device context pointer
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*plaintext*	Input plaintext to be encrypted (16 bytes).
out	*ciphertext*	Output ciphertext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.37 atcab_aes_gcm_aad_update()

```
ATCA_STATUS atcab_aes_gcm_aad_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * aad,
            uint32_t aad_size )
```

Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.

This can be called multiple times. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before atcab_aes_gcm_encrypt_update() or atcab_aes_gcm_decrypt_update().

Parameters

in	*ctx*	AES GCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.38 atcab_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_decrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * tag,
            size_t tag_size,
            bool * is_verified )
```

Complete a GCM decrypt operation verifying the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
in	*tag*	Expected authentication tag.
in	*tag_size*	Size of tag in bytes (12 to 16 bytes).
out	*is_verified*	Returns whether or not the tag verified.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.39 atcab_aes_gcm_decrypt_update()

```
ATCA_STATUS atcab_aes_gcm_decrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*ciphertext*	Ciphertext to be decrypted.
in	*ciphertext_size*	Size of ciphertext in bytes.
out	*plaintext*	Decrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.40 atcab_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_encrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
            uint8_t * tag,
            size_t tag_size )
```

Complete a GCM encrypt operation returning the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
out	*tag*	Authentication tag is returned here.
in	*tag_size*	Tag size in bytes (12 to 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.41 atcab_aes_gcm_encrypt_update()

```
ATCA_STATUS atcab_aes_gcm_encrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
in	*plaintext_size*	Size of plaintext in bytes.
out	*ciphertext*	Encrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.42 atcab_aes_gcm_init()

```
ATCA_STATUS atcab_aes_gcm_init (
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv,
            size_t iv_size )
```

Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*ctx*	AES GCM context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector.
in	*iv_size*	Size of IV in bytes. Standard is 12 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.43 atcab_aes_gcm_init_rand()

```
ATCA_STATUS atcab_aes_gcm_init_rand (
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            size_t rand_size,
            const uint8_t * free_field,
            size_t free_field_size,
            uint8_t * iv )
```

Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*rand_size*	Size of the random field in bytes. Minimum and recommended size is 12 bytes. Max is 32 bytes.
in	*free_field*	Fixed data to include in the IV after the random field. Can be NULL if not used.
in	*free_field_size*	Size of the free field in bytes.
out	*iv*	Initialization vector is returned here. Its size will be rand_size and free_field_size combined.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.44 atcab_aes_gfm()

```
ATCA_STATUS atcab_aes_gfm (
            const uint8_t * h,
            const uint8_t * input,
            uint8_t * output )
```

Perform a Galois Field Multiply (GFM) operation.

Parameters

in	*h*	First input value (16 bytes).
in	*input*	Second input value (16 bytes).
out	*output*	GFM result is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.45 atcab_challenge()

```
ATCA_STATUS atcab_challenge (
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.46 atcab_challenge_seed_update()

```
ATCA_STATUS atcab_challenge_seed_update (
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.

Parameters

in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.47 atcab_checkmac()

```
ATCA_STATUS atcab_checkmac (
            uint8_t mode,
```

```
              uint16_t key_id,
              const uint8_t * challenge,
              const uint8_t * response,
              const uint8_t * other_data )
```

Compares a MAC response with input values.

Parameters

in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key location in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge data (32 bytes)
in	*response*	MAC response data (32 bytes)
in	*other_data*	OtherData parameter (13 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.48 atcab_cmp_config_zone()

```
ATCA_STATUS atcab_cmp_config_zone (
              uint8_t * config_data,
              bool * same_config )
```

Compares a specified configuration zone with the configuration zone currently on the device.

This only compares the static portions of the configuration zone and skips those that are unique per device (first 16 bytes) and areas that can change after the configuration zone has been locked (e.g. LastKeyUse).

Parameters

in	*config_data*	Full configuration data to compare the device against.
out	*same_config*	Result is returned here. True if the static portions on the configuration zones are the same.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.49 atcab_counter()

```
ATCA_STATUS atcab_counter (
              uint8_t mode,
              uint16_t counter_id,
              uint32_t * counter_value )
```

Compute the Counter functions.

Parameters

in	*mode*	the mode used for the counter
in	*counter_id*	The counter to be used
out	*counter_value*	pointer to the counter value returned from device

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.50 atcab_counter_increment()

```
ATCA_STATUS atcab_counter_increment (
            uint16_t counter_id,
            uint32_t * counter_value )
```

Increments one of the device's monotonic counters.

Parameters

in	*counter_id*	Counter to be incremented
out	*counter_value*	New value of the counter is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.51 atcab_counter_read()

```
ATCA_STATUS atcab_counter_read (
            uint16_t counter_id,
            uint32_t * counter_value )
```

Read one of the device's monotonic counters.

Parameters

in	*counter_id*	Counter to be read
out	*counter_value*	Counter value is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.52 atcab_derivekey()

```
ATCA_STATUS atcab_derivekey (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * mac )
```

Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.

Parameters

in	*mode*	Bit 2 must match the value in TempKey.SourceFlag
in	*key← _id*	Key slot to be written
in	*mac*	Optional 32 byte MAC used to validate operation. NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.53 atcab_ecdh()

```
ATCA_STATUS atcab_ecdh (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in a slot and the premaster secret is returned in the clear.

Parameters

in	*key_id*	Slot of private key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here. 32 bytes.

Returns

ATCA_SUCCESS on success

8.1.4.54 atcab_ecdh_base()

```
ATCA_STATUS atcab_ecdh_base (
            uint8_t mode,
            uint16_t key_id,
```

```
const uint8_t * public_key,
uint8_t * pms,
uint8_t * out_nonce )
```

Base function for generating premaster secret key using ECDH.

Parameters

in	*mode*	Mode to be used for ECDH computation
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH pre-master secret is returned here (32 bytes) if returned directly. Otherwise NULL.
out	*out_nonce*	Nonce used to encrypt pre-master secret. NULL if output encryption not used.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.55 atcab_ecdh_enc()

```
ATCA_STATUS atcab_ecdh_enc (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * read_key,
            uint16_t read_key_id,
            const uint8_t num_in[(20)] )
```

ECDH command with a private key in a slot and the premaster secret is read from the next slot.

This function only works for even numbered slots with the proper configuration.

Parameters

in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*read_key*	Read key for the premaster secret slot (key_id|1).
in	*read_key_id*	Read key slot for read_key.
in	*num_in*	20 byte host nonce to inject into Nonce calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.56 atcab_ecdh_ioenc()

```
ATCA_STATUS atcab_ecdh_ioenc (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.57 atcab_ecdh_tempkey()

```
ATCA_STATUS atcab_ecdh_tempkey (
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in TempKey and the premaster secret is returned in the clear.

Parameters

in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.58 atcab_ecdh_tempkey_ioenc()

```
ATCA_STATUS atcab_ecdh_tempkey_ioenc (
            const uint8_t * public_key,
```

```
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.59 atcab_gendig()

```
ATCA_STATUS atcab_gendig (
            uint8_t zone,
            uint16_t key_id,
            const uint8_t * other_data,
            uint8_t other_data_size )
```

Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.

Parameters

in	*zone*	Designates the source of the data to hash with TempKey.
in	*key_id*	Indicates the key, OTP block, or message order for shared nonce mode.
in	*other_data*	Four bytes of data for SHA calculation when using a NoMac key, 32 bytes for "Shared Nonce" mode, otherwise ignored (can be NULL).
in	*other_data_size*	Size of other_data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.60 atcab_genkey()

```
ATCA_STATUS atcab_genkey (
            uint16_t key_id,
            uint8_t * public_key )
```

Issues GenKey command, which generates a new random private key in slot/handle and returns the public key.

Parameters

in	*key_id*	Slot number where an ECC private key is configured. Can also be ATCA_TEMPKEY_KEYID to generate a private key in TempKey.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.61 atcab_genkey_base()

```
ATCA_STATUS atcab_genkey_base (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * other_data,
            uint8_t * public_key )
```

Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.

Parameters

in	*mode*	Mode determines what operations the GenKey command performs.
in	*key_id*	Slot to perform the GenKey command on.
in	*other_data*	OtherData for PubKey digest calculation. Can be set to NULL otherwise.
out	*public_key*	If the mode indicates a public key will be calculated, it will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.62 atcab_get_device()

```
ATCADevice atcab_get_device (
            void )
```

Get the global device object.

Returns

instance of global ATCADevice

8.1.4.63 atcab_get_device_address()

```
uint8_t atcab_get_device_address (
            ATCADevice device )
```

Get the current device address based on the configured device and interface.

Returns

the device address if applicable else 0xFF

8.1.4.64 atcab_get_device_type()

```
ATCADeviceType atcab_get_device_type (
            void )
```

Get the current device type configured for the global ATCADevice.

Returns

Device type if basic api is initialized or ATCA_DEV_UNKNOWN.

8.1.4.65 atcab_get_device_type_ext()

```
ATCADeviceType atcab_get_device_type_ext (
            ATCADevice device )
```

Get the selected device type of rthe device context.

Parameters

in	*device*	Device context pointer

Returns

Device type if basic api is initialized or ATCA_DEV_UNKNOWN.

8.1.4.66 atcab_get_pubkey()

```
ATCA_STATUS atcab_get_pubkey (
            uint16_t key_id,
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from an existing private key in a slot.

Parameters

in	*key_id*	Slot number of the private key.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.67 atcab_get_pubkey_ext()

```
ATCA_STATUS atcab_get_pubkey_ext (
            ATCADevice device,
            uint16_t key_id,
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from an existing private key in a slot.

Parameters

in	*key_id*	Slot number of the private key.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.68 atcab_get_zone_size()

```
ATCA_STATUS atcab_get_zone_size (
            uint8_t zone,
            uint16_t slot,
            size_t * size )
```

Gets the size of the specified zone in bytes.

Parameters

in	*zone*	Zone to get size information from. Config(0), OTP(1), or Data(2) which requires a slot.
in	*slot*	If zone is Data(2), the slot to query for size.
out	*size*	Zone size is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.69 atcab_hmac()

```
ATCA_STATUS atcab_hmac (
            uint8_t mode,
            uint16_t key_id,
            uint8_t * digest )
```

Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*mode*	Controls which fields within the device are used in the message.
in	*key_id*	Which key is to be used to generate the response. Bits 0:3 only are used to select a slot but all 16 bits are used in the HMAC message.
out	*digest*	HMAC digest is returned in this buffer (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.70 atcab_hw_sha2_256()

```
ATCA_STATUS atcab_hw_sha2_256 (
            const uint8_t * data,
            size_t data_size,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.71 atcab_hw_sha2_256_finish()

```
ATCA_STATUS atcab_hw_sha2_256_finish (
            atca_sha256_ctx_t * ctx,
            uint8_t * digest )
```

Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*ctx*	SHA256 context
out	*digest*	SHA256 digest is returned here (32 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.72 atcab_hw_sha2_256_init()

```
ATCA_STATUS atcab_hw_sha2_256_init (
            atca_sha256_ctx_t * ctx )
```

Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.

Parameters

in	*ctx*	SHA256 context

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.73 atcab_hw_sha2_256_update()

```
ATCA_STATUS atcab_hw_sha2_256_update (
            atca_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add message data to a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*ctx*	SHA256 context
in	*data*	Message data to be added to hash.
in	*data_size*	Size of data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.74 atcab_idle()

```
ATCA_STATUS atcab_idle (
            void )
```

idle the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.75 atcab_info()

```
ATCA_STATUS atcab_info (
            uint8_t * revision )
```

Use the Info command to get the device revision (DevRev).

Parameters

out	*revision*	Device revision is returned here (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.76 atcab_info_base()

```
ATCA_STATUS atcab_info_base (
            uint8_t mode,
            uint16_t param2,
            uint8_t * out_data )
```

Issues an Info command, which return internal device information and can control GPIO and the persistent latch.

Parameters

in	*mode*	Selects which mode to be used for info command.
in	*param2*	Selects the particular fields for the mode.
out	*out_data*	Response from info command (4 bytes). Can be set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.77 atcab_info_get_latch()

```
ATCA_STATUS atcab_info_get_latch (
            bool * state )
```

Use the Info command to get the persistent latch current state for an ATECC608 device.

Parameters

out	*state*	The state is returned here. Set (true) or Cler (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.78 atcab_info_set_latch()

```
ATCA_STATUS atcab_info_set_latch (
            bool state )
```

Use the Info command to set the persistent latch state for an ATECC608 device.

Parameters

out	*state*	Persistent latch state. Set (true) or clear (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.79 atcab_init()

```
ATCA_STATUS atcab_init (
            ATCAIfaceCfg * cfg )
```

Creates a global ATCADevice object used by Basic API.

Parameters

in	*cfg*	Logical interface configuration. Some predefined configurations can be found in atca_cfgs.h

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.80 atcab_init_device()

```
ATCA_STATUS atcab_init_device (
            ATCADevice ca_device )
```

Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.

Parameters

in	*ca_device*	ATCADevice instance to use as the global Basic API crypto device instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.81 atcab_init_ext()

```
ATCA_STATUS atcab_init_ext (
            ATCADevice * device,
            ATCAIfaceCfg * cfg )
```

Creates and initializes a ATCADevice context.

Parameters

out	*device*	Pointer to the device context pointer
in	*cfg*	Logical interface configuration. Some predefined configurations can be found in atca_cfgs.h

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.82 atcab_is_ca_device()

```
bool atcab_is_ca_device (
            ATCADeviceType dev_type )
```

Check whether the device is cryptoauth device.

Returns

True if device is cryptoauth device or False.

8.1.4.83 atcab_is_config_locked()

```
ATCA_STATUS atcab_is_config_locked (
            bool * is_locked )
```

This function check whether configuration zone is locked or not.

Parameters

out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.84 atcab_is_data_locked()

```
ATCA_STATUS atcab_is_data_locked (
            bool * is_locked )
```

This function check whether data/setup zone is locked or not.

Parameters

out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.85 atcab_is_locked()

```
ATCA_STATUS atcab_is_locked (
            uint8_t zone,
            bool * is_locked )
```

Executes Read command, which reads the configuration zone to see if the specified zone is locked.

Parameters

in	*zone*	The zone to query for locked (use LOCK_ZONE_CONFIG or LOCK_ZONE_DATA).
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.86 atcab_is_private()

```
ATCA_STATUS atcab_is_private (
            uint16_t slot,
            bool * is_private )
```

8.1.4.87 atcab_is_private_ext()

```
ATCA_STATUS atcab_is_private_ext (
            ATCADevice device,
            uint16_t slot,
            bool * is_private )
```

Check to see if the key is a private key or not.

This function will issue the Read command as many times as is required to read the requested data.

Parameters

in	*slot*	Slot number to read from if zone is ATCA_ZONE_DATA(2). Ignored for all other zones.
out	*is_private*	Returned valud if successful. True if key is private.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.88 atcab_is_slot_locked()

```
ATCA_STATUS atcab_is_slot_locked (
            uint16_t slot,
            bool * is_locked )
```

This function check whether slot/handle is locked or not.

Parameters

in	*slot*	Slot to query for locked
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.89 atcab_is_ta_device()

```
bool atcab_is_ta_device (
            ATCADeviceType dev_type )
```

Check whether the device is Trust Anchor device.

Returns

True if device is Trust Anchor device or False.

8.1.4.90 atcab_kdf()

```
ATCA_STATUS atcab_kdf (
            uint8_t mode,
            uint16_t key_id,
            const uint32_t details,
            const uint8_t * message,
            uint8_t * out_data,
            uint8_t * out_nonce )
```

Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.

Generally this function combines a source key with an input string and creates a result key/digest/array.

Parameters

in	*mode*	Mode determines KDF algorithm (PRF,AES,HKDF), source key location, and target key locations.
in	*key_id*	Source and target key slots if locations are in the EEPROM. Source key slot is the LSB and target key slot is the MSB.
in	*details*	Further information about the computation, depending on the algorithm (4 bytes).
in	*message*	Input value from system (up to 128 bytes). Actual size of message is 16 bytes for AES algorithm or is encoded in the MSB of the details parameter for other algorithms.
out	*out_data*	Output of the KDF function is returned here. If the result remains in the device, this can

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.91 atcab_lock()

```
ATCA_STATUS atcab_lock (
            uint8_t mode,
            uint16_t summary_crc )
```

The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.

Parameters

in	*mode*	Zone, and/or slot, and summary check (bit 7).
in	*summary_crc*	CRC of the config or data zones. Ignored for slot locks or when mode bit 7 is set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.92 atcab_lock_config_zone()

```
ATCA_STATUS atcab_lock_config_zone (
            void )
```

Unconditionally (no CRC required) lock the config zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.93 atcab_lock_config_zone_crc()

```
ATCA_STATUS atcab_lock_config_zone_crc (
            uint16_t summary_crc )
```

Lock the config zone with summary CRC.

The CRC is calculated over the entire config zone contents. 48 bytes for TA100, 88 bytes for ATSHA devices, 128 bytes for ATECC devices. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*summary_crc*	Expected CRC over the config zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.94 atcab_lock_data_slot()

```
ATCA_STATUS atcab_lock_data_slot (
            uint16_t slot )
```

Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1) (for cryptoauth devices) or Lock an individual handle in shared data element on an Trust Anchor device (for Trust Anchor devices).

Parameters

in	*slot*	Slot to be locked in data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.95 atcab_lock_data_zone()

```
ATCA_STATUS atcab_lock_data_zone (
            void )
```

Unconditionally (no CRC required) lock the data zone (slots and OTP). for CryptoAuth devices and lock the setup for Trust Anchor device.

ConfigZone must be locked and DataZone must be unlocked for the zone to be successfully locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.96 atcab_lock_data_zone_crc()

```
ATCA_STATUS atcab_lock_data_zone_crc (
            uint16_t summary_crc )
```

Lock the data zone (slots and OTP) with summary CRC.

The CRC is calculated over the concatenated contents of all the slots and OTP at the end. Private keys (Key↩Config.Private=1) are skipped. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*summary_crc*	Expected CRC over the data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.97 atcab_mac()

```
ATCA_STATUS atcab_mac (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * challenge,
            uint8_t * digest )
```

Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge message (32 bytes). May be NULL if mode indicates a challenge isn't required.
out	*digest*	MAC response is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.98 atcab_nonce()

```
ATCA_STATUS atcab_nonce (
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.99 atcab_nonce_base()

```
ATCA_STATUS atcab_nonce_base (
            uint8_t mode,
            uint16_t zero,
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.

Parameters

in	*mode*	Controls the mechanism of the internal RNG or fixed write.
in	*zero*	Param2, normally 0, but can be used to indicate a nonce calculation mode (bit 15).
in	*num_in*	Input value to either be included in the nonce calculation in random modes (20 bytes) or to be written directly (32 bytes or 64 bytes(ATECC608)) in pass-through mode.
out	*rand_out*	If using a random mode, the internally generated 32-byte random number that was used in the nonce calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.100 atcab_nonce_load()

```
ATCA_STATUS atcab_nonce_load (
            uint8_t target,
            const uint8_t * num_in,
            uint16_t num_in_size )
```

Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.

For the ATECC608, available targets are TempKey (32 or 64 bytes), Message Digest Buffer (32 or 64 bytes), or the Alternate Key Buffer (32 bytes). For all other devices, only TempKey (32 bytes) is available.

Parameters

in	*target*	Target device buffer to load. Can be NONCE_MODE_TARGET_TEMPKEY, NONCE_MODE_TARGET_MSGDIGBUF, or NONCE_MODE_TARGET_ALTKEYBUF.
in	*num_in*	Data to load into the buffer.
in	*num_in_size*	Size of num_in in bytes. Can be 32 or 64 bytes depending on device and target.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.101 atcab_nonce_rand()

```
ATCA_STATUS atcab_nonce_rand (
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.

Parameters

in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.102 atcab_pbkdf2_sha256()

```
ATCA_STATUS atcab_pbkdf2_sha256 (
            const uint32_t iter,
            const uint16_t slot,
            const uint8_t * salt,
            const size_t salt_len,
            uint8_t * result,
            size_t result_len )
```

Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iter*	Number of iterations of the algorithm to perform
in	*slot*	Slot/handle with a stored key (password)
in	*salt*	Salt bytes to use
in	*salt_len*	Length of the salt bytes buffer
out	*result*	Output buffer to hold the derived key
in	*result_len*	Length of the key to derive

8.1.4.103 atcab_pbkdf2_sha256_ext()

```
ATCA_STATUS atcab_pbkdf2_sha256_ext (
             ATCADevice device,
             const uint32_t iter,
             const uint16_t slot,
             const uint8_t * salt,
             const size_t salt_len,
             uint8_t * result,
             size_t result_len )
```

Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*device*	Device context pointer
in	*iter*	Number of iterations of the algorithm to perform
in	*slot*	Slot/handle with a stored key (password)
in	*salt*	Salt bytes to use
in	*salt_len*	Length of the salt bytes buffer
out	*result*	Output buffer to hold the derived key
in	*result_len*	Length of the key to derive

8.1.4.104 atcab_printbin()

```
ATCA_STATUS atcab_printbin (
             uint8_t * binary,
             size_t bin_len,
             bool add_space )
```

8.1.4.105 atcab_priv_write()

```
ATCA_STATUS atcab_priv_write (
             uint16_t key_id,
             const uint8_t priv_key[36],
             uint16_t write_key_id,
             const uint8_t write_key[32],
             const uint8_t num_in[(20)] )
```

Executes PrivWrite command, to write externally generated ECC private keys into the device.

Parameters

in	*key_id*	Slot to write the external private key into.
in	*priv_key*	External private key (36 bytes) to be written. The first 4 bytes should be zero for P256 curve.
in	*write_key_id*	Write key slot. Ignored if write_key is NULL.
in	*write_key*	Write key (32 bytes). If NULL, perform an unencrypted PrivWrite, which is only available when the data zone is unlocked.
in	*num_in*	20 byte host nonce to inject into Nonce calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.106 atcab_random()

```
ATCA_STATUS atcab_random (
            uint8_t * rand_out )
```

Executes Random command, which generates a 32 byte random number from the device.

Parameters

out	*rand_out*	32 bytes of random data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.107 atcab_random_ext()

```
ATCA_STATUS atcab_random_ext (
            ATCADevice device,
            uint8_t * rand_out )
```

Executes Random command, which generates a 32 byte random number from the device.

Parameters

in	*device*	Device context pointer
out	*rand_out*	32 bytes of random data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.108 atcab_read_bytes_zone()

```
ATCA_STATUS atcab_read_bytes_zone (
            uint8_t zone,
            uint16_t slot,
            size_t offset,
            uint8_t * data,
            size_t length )
```

Used to read an arbitrary number of bytes from any zone configured for clear reads.

This function will issue the Read command as many times as is required to read the requested data.

Parameters

in	*zone*	Zone to read data from. Option are ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	Slot number to read from if zone is ATCA_ZONE_DATA(2). Ignored for all other zones.
in	*offset*	Byte offset within the zone to read from.
out	*data*	Read data is returned here.
in	*length*	Number of bytes to read starting from the offset.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.109 atcab_read_config_zone()

```
ATCA_STATUS atcab_read_config_zone (
            uint8_t * config_data )
```

Executes Read command to read the complete device configuration zone.

Parameters

out	*config_data*	Configuration zone data is returned here. 88 bytes for ATSHA devices, 128 bytes for ATECC devices and 48 bytes for Trust Anchor devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.110 atcab_read_enc()

```
ATCA_STATUS atcab_read_enc (
            uint16_t key_id,
            uint8_t block,
            uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[(20)] )
```

Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.

Data zone must be locked for this command to succeed. Can only read 32 byte blocks.

Parameters

in	*key_id*	The slot ID to read from.
in	*block*	Index of the 32 byte block within the slot to read.
out	*data*	Decrypted (plaintext) data from the read is returned here (32 bytes).
in	*enc_key*	32 byte ReadKey for the slot being read.
in	*enc_key_id*	KeyID of the ReadKey being used.
in	*num_in*	20 byte host nonce to inject into Nonce calculation

returns ATCA_SUCCESS on success, otherwise an error code.

8.1.4.111 atcab_read_pubkey()

```
ATCA_STATUS atcab_read_pubkey (
            uint16_t slot,
            uint8_t * public_key )
```

Executes Read command to read an ECC P256 public key from a slot configured for clear reads.

This function assumes the public key is stored using the ECC public key format specified in the datasheet.

Parameters

in	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a public key.
out	*public_key*	Public key is returned here (64 bytes). Format will be the 32 byte X and Y big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.112 atcab_read_pubkey_ext()

```
ATCA_STATUS atcab_read_pubkey_ext (
            ATCADevice device,
```

```
            uint16_t slot,
            uint8_t * public_key )
```

Executes Read command to read an ECC P256 public key from a slot configured for clear reads.

This function assumes the public key is stored using the ECC public key format specified in the datasheet.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a public key.
out	*public_key*	Public key is returned here (64 bytes). Format will be the 32 byte X and Y big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.113 atcab_read_serial_number()

```
ATCA_STATUS atcab_read_serial_number (
            uint8_t * serial_number )
```

This function returns serial number of the device.

Parameters

out	*serial_number*	9 byte serial number is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.114 atcab_read_sig()

```
ATCA_STATUS atcab_read_sig (
            uint16_t slot,
            uint8_t * sig )
```

Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.

Parameters

in	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a signature.
out	*sig*	Signature will be returned here (64 bytes). Format will be the 32 byte R and S big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.115 atcab_read_zone()

```
ATCA_STATUS atcab_read_zone (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            uint8_t * data,
            uint8_t len )
```

Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.

When reading a slot or OTP, data zone must be locked and the slot configuration must not be secret for a slot to be successfully read.

Parameters

in	*zone*	Zone to be read from device. Options are ATCA_ZONE_CONFIG, ATCA_ZONE_OTP, or ATCA_ZONE_DATA.
in	*slot*	Slot number for data zone and ignored for other zones.
in	*block*	32 byte block index within the zone.
in	*offset*	4 byte work index within the block. Ignored for 32 byte reads.
out	*data*	Read data is returned here.
in	*len*	Length of the data to be read. Must be either 4 or 32.

returns ATCA_SUCCESS on success, otherwise an error code.

8.1.4.116 atcab_release()

```
ATCA_STATUS atcab_release (
            void )
```

release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.

Returns

Returns ATCA_SUCCESS .

8.1.4.117 atcab_release_ext()

```
ATCA_STATUS atcab_release_ext (
            ATCADevice * device )
```

release (free) the an ATCADevice instance.

Parameters

in	*device*	Pointer to the device context pointer

Returns

Returns ATCA_SUCCESS .

8.1.4.118 atcab_secureboot()

```
ATCA_STATUS atcab_secureboot (
            uint8_t mode,
            uint16_t param2,
            const uint8_t * digest,
            const uint8_t * signature,
            uint8_t * mac )
```

Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.

Parameters

in	*mode*	Mode determines what operations the SecureBoot command performs.
in	*param2*	Not used, must be 0.
in	*digest*	Digest of the code to be verified (32 bytes).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
out	*mac*	Validating MAC will be returned here (32 bytes). Can be NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.119 atcab_secureboot_mac()

```
ATCA_STATUS atcab_secureboot_mac (
            uint8_t mode,
            const uint8_t * digest,
            const uint8_t * signature,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.

Parameters

in	*mode*	Mode determines what operations the SecureBoot command performs.

Parameters

in	*digest*	Digest of the code to be verified (32 bytes). This is the plaintext digest (not encrypted).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
in	*num_in*	Host nonce (20 bytes).
in	*io_key*	IO protection key (32 bytes).
out	*is_verified*	Verify result is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.120 atcab_selftest()

```
ATCA_STATUS atcab_selftest (
            uint8_t mode,
            uint16_t param2,
            uint8_t * result )
```

Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608 chip.

Parameters

in	*mode*	Functions to test. Can be a bit field combining any of the following: SELFTEST_MODE_RNG, SELFTEST_MODE_ECDSA_VERIFY, SELFTEST_MODE_ECDSA_SIGN, SELFTEST_MODE_ECDH, SELFTEST_MODE_AES, SELFTEST_MODE_SHA, SELFTEST_MODE_ALL.
in	*param2*	Currently unused, should be 0.
out	*result*	Results are returned here as a bit field.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.121 atcab_sha()

```
ATCA_STATUS atcab_sha (
            uint16_t length,
            const uint8_t * message,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*length*	Size of message parameter in bytes.
in	*message*	Message data to be hashed.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.122 atcab_sha_base()

```
ATCA_STATUS atcab_sha_base (
            uint8_t mode,
            uint16_t length,
            const uint8_t * data_in,
            uint8_t * data_out,
            uint16_t * data_out_size )
```

Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.

Only the Start(0) and Compute(1) modes are available for ATSHA devices.

Parameters

in	*mode*	SHA command mode Start(0), Update/Compute(1), End(2), Public(3), HMACstart(4), HMACend(5), Read_Context(6), or Write_Context(7). Also message digest target location for the ATECC608.
in	*length*	Number of bytes in the message parameter or KeySlot for the HMAC key if Mode is HMACstart(4) or Public(3).
in	*data_in*	Message bytes to be hashed or Write_Context if restoring a context on the ATECC608. Can be NULL if not required by the mode.
out	*data_out*	Data returned by the command (digest or context).
in,out	*data_out_size*	As input, the size of the data_out buffer. As output, the number of bytes returned in data_out.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.123 atcab_sha_end()

```
ATCA_STATUS atcab_sha_end (
            uint8_t * digest,
```

```
uint16_t length,
const uint8_t * message )
```

Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.

Parameters

out	*digest*	Digest from SHA-256 or HMAC/SHA-256 will be returned here (32 bytes).
in	*length*	Length of any remaining data to include in hash. Max 64 bytes.
in	*message*	Remaining data to include in hash. NULL if length is 0.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.124 atcab_sha_hmac()

```
ATCA_STATUS atcab_sha_hmac (
            const uint8_t * data,
            size_t data_size,
            uint16_t key_slot,
            uint8_t * digest,
            uint8_t target )
```

Use the SHA command to compute an HMAC/SHA-256 operation.

Parameters

in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
in	*key_slot*	Slot key id to use for the HMAC calculation
out	*digest*	Digest is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.125 atcab_sha_hmac_ext()

```
ATCA_STATUS atcab_sha_hmac_ext (
            ATCADevice device,
            const uint8_t * data,
            size_t data_size,
            uint16_t key_slot,
            uint8_t * digest,
            uint8_t target )
```

Use the SHA command to compute an HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
in	*key_slot*	Slot key id to use for the HMAC calculation
out	*digest*	Digest is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.126 atcab_sha_hmac_finish()

```
ATCA_STATUS atcab_sha_hmac_finish (
            atca_hmac_sha256_ctx_t * ctx,
            uint8_t * digest,
            uint8_t target )
```

Executes SHA command to complete a HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
out	*digest*	HMAC/SHA-256 result is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.127 atcab_sha_hmac_init()

```
ATCA_STATUS atcab_sha_hmac_init (
            atca_hmac_sha256_ctx_t * ctx,
            uint16_t key_slot )
```

Executes SHA command to start an HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
in	*key_slot*	Slot key id to use for the HMAC calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.128 atcab_sha_hmac_update()

```
ATCA_STATUS atcab_sha_hmac_update (
            atca_hmac_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
in	*data*	Message data to add
in	*data_size*	Size of message data in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.129 atcab_sha_read_context()

```
ATCA_STATUS atcab_sha_read_context (
            uint8_t * context,
            uint16_t * context_size )
```

Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.

Parameters

out	*context*	Context data is returned here.
in,out	*context_size*	As input, the size of the context buffer in bytes. As output, the size of the returned context data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.130 atcab_sha_start()

```
ATCA_STATUS atcab_sha_start (
            void )
```

Executes SHA command to initialize SHA-256 calculation engine.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.131 atcab_sha_update()

```
ATCA_STATUS atcab_sha_update (
            const uint8_t * message )
```

Executes SHA command to add 64 bytes of message data to the current context.

Parameters

in	*message*	64 bytes of message data to add to add to operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.132 atcab_sha_write_context()

```
ATCA_STATUS atcab_sha_write_context (
            const uint8_t * context,
            uint16_t context_size )
```

Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.

Parameters

in	*context*	Context data to be restored.
in	*context_size*	Size of the context data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.133 atcab_sign()

```
ATCA_STATUS atcab_sign (
            uint16_t key_id,
            const uint8_t * msg,
            uint8_t * signature )
```

Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*key_id*	Slot of the private key to be used to sign the message.
in	*msg*	32-byte message to be signed. Typically the SHA256 hash of the full message.
out	*signature*	Signature will be returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.134 atcab_sign_base()

```
ATCA_STATUS atcab_sign_base (
            uint8_t mode,
            uint16_t key_id,
            uint8_t * signature )
```

Executes the Sign command, which generates a signature using the ECDSA algorithm.

Parameters

in	*mode*	Mode determines what the source of the message to be signed.
in	*key_id*	Private key slot used to sign the message.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.135 atcab_sign_ext()

```
ATCA_STATUS atcab_sign_ext (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * msg,
            uint8_t * signature )
```

Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of the private key to be used to sign the message.
in	*msg*	32-byte message to be signed. Typically the SHA256 hash of the full message.
out	*signature*	Signature will be returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.136 atcab_sign_internal()

```
ATCA_STATUS atcab_sign_internal (
            uint16_t key_id,
            bool is_invalidate,
            bool is_full_sn,
            uint8_t * signature )
```

Executes Sign command to sign an internally generated message.

Parameters

in	*key_id*	Slot of the private key to be used to sign the message.
in	*is_invalidate*	Set to true if the signature will be used with the Verify(Invalidate) command. false for all other cases.
in	*is_full_sn*	Set to true if the message should incorporate the device's full serial number.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.137 atcab_sleep()

```
ATCA_STATUS atcab_sleep (
            void )
```

invoke sleep on the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.138 atcab_updateextra()

```
ATCA_STATUS atcab_updateextra (
            uint8_t mode,
            uint16_t new_value )
```

Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

Can also be used to decrement the limited use counter associated with the key in slot NewValue.

Parameters

in	*mode*	Mode determines what operations the UpdateExtra command performs.
in	*new_value*	Value to be written.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.139 atcab_verify()

```
ATCA_STATUS atcab_verify (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * public_key,
            const uint8_t * other_data,
            uint8_t * mac )
```

Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.

For the Stored, External, and ValidateExternal Modes, the contents of TempKey (or Message Digest Buffer in some cases for the ATECC608) should contain the 32 byte message.

Parameters

in	*mode*	Verify command mode and options
in	*key_id*	Stored mode, the slot containing the public key to be used for the verification. ValidateExternal mode, the slot containing the public key to be validated. External mode, KeyID contains the curve type to be used to Verify the signature. Validate or Invalidate mode, the slot containing the public key to be (in)validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	If mode is External, the public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve. NULL for all other modes.
in	*other_data*	If mode is Validate, the bytes used to generate the message for the validation (19 bytes). NULL for all other modes.
out	*mac*	If mode indicates a validating MAC, then the MAC will will be returned here. Can be NULL otherwise.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.140 atcab_verify_extern()

```
ATCA_STATUS atcab_verify_extern (
            const uint8_t * message,
            const uint8_t * signature,
            const uint8_t * public_key,
            bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.141 atcab_verify_extern_ext()

```
ATCA_STATUS atcab_verify_extern_ext (
            ATCADevice device,
```

```
const uint8_t * message,
const uint8_t * signature,
const uint8_t * public_key,
bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.142 atcab_verify_extern_mac()

```
ATCA_STATUS atcab_verify_extern_mac (
        const uint8_t * message,
        const uint8_t * signature,
        const uint8_t * public_key,
        const uint8_t * num_in,
        const uint8_t * io_key,
        bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.143 atcab_verify_invalidate()

```
ATCA_STATUS atcab_verify_invalidate (
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * other_data,
            bool * is_verified )
```

Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be invalidated in TempKey (mode=0x10).

Parameters

in	*key_id*	Slot containing the public key to be invalidated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.144 atcab_verify_stored()

```
ATCA_STATUS atcab_verify_stored (
            const uint8_t * message,
            const uint8_t * signature,
            uint16_t key_id,
            bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.145 atcab_verify_stored_ext()

```
ATCA_STATUS atcab_verify_stored_ext (
            ATCADevice device,
            const uint8_t * message,
            const uint8_t * signature,
            uint16_t key_id,
            bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.146 atcab_verify_stored_mac()

```
ATCA_STATUS atcab_verify_stored_mac (
            const uint8_t * message,
            const uint8_t * signature,
            uint16_t key_id,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.147 atcab_verify_validate()

```
ATCA_STATUS atcab_verify_validate (
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * other_data,
            bool * is_verified )
```

Executes the Verify command in Validate mode to validate a public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be validated in TempKey (mode=0x10).

Parameters

in	*key_id*	Slot containing the public key to be validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.1.4.148 atcab_version()

```
ATCA_STATUS atcab_version (
            char * ver_str )
```

basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.

returns a version string for the CryptoAuthLib release. The format of the version string returned is "yyyymmdd"

Parameters

out	*ver_str*	ptr to space to receive version string

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.149 atcab_wakeup()

```
ATCA_STATUS atcab_wakeup (
            void )
```

wakeup the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.150 atcab_write()

```
ATCA_STATUS atcab_write (
            uint8_t zone,
            uint16_t address,
            const uint8_t * value,
            const uint8_t * mac )
```

Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.

Parameters

in	*zone*	Zone/Param1 for the write command.
in	*address*	Address/Param2 for the write command.
in	*value*	Plain-text data to be written or cipher-text for encrypted writes. 32 or 4 bytes depending on bit 7 in the zone.
in	*mac*	MAC required for encrypted writes (32 bytes). Set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.151 atcab_write_bytes_zone()

```
ATCA_STATUS atcab_write_bytes_zone (
            uint8_t zone,
            uint16_t slot,
            size_t offset_bytes,
```

```
            const uint8_t * data,
            size_t length )
```

Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).

Config zone must be unlocked for writes to that zone. If data zone is unlocked, only 32-byte writes are allowed to slots and OTP and the offset and length must be multiples of 32 or the write will fail.

Parameters

in	*zone*	Zone to write data to: ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	If zone is ATCA_ZONE_DATA(2), the slot number to write to. Ignored for all other zones.
in	*offset_bytes*	Byte offset within the zone to write to. Must be a multiple of a word (4 bytes).
in	*data*	Data to be written.
in	*length*	Number of bytes to be written. Must be a multiple of a word (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.152 atcab_write_config_counter()

```
ATCA_STATUS atcab_write_config_counter (
            uint16_t counter_id,
            uint32_t counter_value )
```

Initialize one of the monotonic counters in device with a specific value.

The monotonic counters are stored in the configuration zone using a special format. This encodes a binary count value into the 8 byte encoded value required. Can only be set while the configuration zone is unlocked.

Parameters

in	*counter_id*	Counter to be written.
in	*counter_value*	Counter value to set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.153 atcab_write_config_zone()

```
ATCA_STATUS atcab_write_config_zone (
            const uint8_t * config_data )
```

Executes the Write command, which writes the configuration zone.

First 16 bytes are skipped as they are not writable. LockValue and LockConfig are also skipped and can only be changed via the Lock command.

This command may fail if UserExtra and/or Selector bytes have already been set to non-zero values.

Parameters

in	*config_data*	Data to the config zone data. This should be 88 bytes for SHA devices and 128 bytes for ECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.154 atcab_write_enc()

```
ATCA_STATUS atcab_write_enc (
            uint16_t key_id,
            uint8_t block,
            const uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[(20)] )
```

Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.

The function takes clear text bytes and encrypts them for writing over the wire. Data zone must be locked and the slot configuration must be set to encrypted write for the block to be successfully written.

Parameters

in	*key_id*	Slot ID to write to.
in	*block*	Index of the 32 byte block to write in the slot.
in	*data*	32 bytes of clear text data to be written to the slot
in	*enc_key*	WriteKey to encrypt with for writing
in	*enc_key_id*	The KeyID of the WriteKey
in	*num_in*	20 byte host nonce to inject into Nonce calculation

returns ATCA_SUCCESS on success, otherwise an error code.

8.1.4.155 atcab_write_pubkey()

```
ATCA_STATUS atcab_write_pubkey (
            uint16_t slot,
            const uint8_t * public_key )
```

Uses the write command to write a public key to a slot in the proper format.

Parameters

in	*slot*	Slot number to write. Only slots 8 to 15 are large enough to store a public key.
in	*public_key*	Public key to write into the slot specified. X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.156 atcab_write_zone()

```
ATCA_STATUS atcab_write_zone (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            const uint8_t * data,
            uint8_t len )
```

Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.

Parameters

in	*zone*	Device zone to write to (0=config, 1=OTP, 2=data).
in	*slot*	If writing to the data zone, it is the slot to write to, otherwise it should be 0.
in	*block*	32-byte block to write to.
in	*offset*	4-byte word within the specified block to write to. If performing a 32-byte write, this should be 0.
in	*data*	Data to be written.
in	*len*	Number of bytes to be written. Must be either 4 or 32.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.157 calib_aes_gcm_aad_update()

```
ATCA_STATUS calib_aes_gcm_aad_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * aad,
            uint32_t aad_size )
```

Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.

This can be called multiple times. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before atcab_aes_gcm_encrypt_update() or atcab_aes_gcm_decrypt_update().

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.158 calib_aes_gcm_decrypt_finish()

```
ATCA_STATUS calib_aes_gcm_decrypt_finish (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * tag,
            size_t tag_size,
            bool * is_verified )
```

Complete a GCM decrypt operation verifying the authentication tag.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*tag*	Expected authentication tag.
in	*tag_size*	Size of tag in bytes (12 to 16 bytes).
out	*is_verified*	Returns whether or not the tag verified.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.159 calib_aes_gcm_decrypt_update()

```
ATCA_STATUS calib_aes_gcm_decrypt_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*ciphertext*	Ciphertext to be decrypted.
in	*ciphertext_size*	Size of ciphertext in bytes.
out	*plaintext*	Decrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.160 calib_aes_gcm_encrypt_finish()

```
ATCA_STATUS calib_aes_gcm_encrypt_finish (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint8_t * tag,
            size_t tag_size )
```

Complete a GCM encrypt operation returning the authentication tag.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
out	*tag*	Authentication tag is returned here.
in	*tag_size*	Tag size in bytes (12 to 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.161 calib_aes_gcm_encrypt_update()

```
ATCA_STATUS calib_aes_gcm_encrypt_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
in	*plaintext_size*	Size of plaintext in bytes.
out	*ciphertext*	Encrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.162 calib_aes_gcm_init()

```
ATCA_STATUS calib_aes_gcm_init (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv,
            size_t iv_size )
```

Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector.
in	*iv_size*	Size of IV in bytes. Standard is 12 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.4.163 calib_aes_gcm_init_rand()

```
ATCA_STATUS calib_aes_gcm_init_rand (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            size_t rand_size,
```

```
const uint8_t * free_field,
size_t free_field_size,
uint8_t * iv )
```

Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*rand_size*	Size of the random field in bytes. Minimum and recommended size is 12 bytes. Max is 32 bytes.
in	*free_field*	Fixed data to include in the IV after the random field. Can be NULL if not used.
in	*free_field_size*	Size of the free field in bytes.
out	*iv*	Initialization vector is returned here. Its size will be rand_size and free_field_size combined.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.1.5 Variable Documentation

8.1.5.1 _gDevice

```
ATCADevice _gDevice [extern]
```

8.1.5.2 atca_basic_aes_gcm_version

```
const char* atca_basic_aes_gcm_version [extern]
```

8.2 Configuration (cfg_)

Logical device configurations describe the CryptoAuth device type and logical interface.

Logical device configurations describe the CryptoAuth device type and logical interface.

8.3 ATCADevice (atca_)

ATCADevice object - composite of command and interface objects.

Data Structures

- struct _atsha204a_config
- struct _atecc508a_config
- struct _atecc608_config
- struct atca_device

  atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods

Macros

- #define ATCA_PACKED
- #define ATCA_AES_ENABLE_EN_SHIFT (0)
- #define ATCA_AES_ENABLE_EN_MASK (0x01u << ATCA_AES_ENABLE_EN_SHIFT)
- #define ATCA_I2C_ENABLE_EN_SHIFT (0)
- #define ATCA_I2C_ENABLE_EN_MASK (0x01u << ATCA_I2C_ENABLE_EN_SHIFT)
- #define ATCA_COUNTER_MATCH_EN_SHIFT (0)
- #define ATCA_COUNTER_MATCH_EN_MASK (0x01u << ATCA_COUNTER_MATCH_EN_SHIFT)
- #define ATCA_COUNTER_MATCH_KEY_SHIFT (4)
- #define ATCA_COUNTER_MATCH_KEY_MASK (0x0Fu << ATCA_COUNTER_MATCH_KEY_SHIFT)
- #define ATCA_COUNTER_MATCH_KEY(v) (ATCA_COUNTER_MATCH_KEY_MASK & (v << ATCA_COUNTER_MATCH_K
- #define ATCA_CHIP_MODE_I2C_EXTRA_SHIFT (0)
- #define ATCA_CHIP_MODE_I2C_EXTRA_MASK (0x01u << ATCA_CHIP_MODE_I2C_EXTRA_SHIFT)
- #define ATCA_CHIP_MODE_TTL_EN_SHIFT (1)
- #define ATCA_CHIP_MODE_TTL_EN_MASK (0x01u << ATCA_CHIP_MODE_TTL_EN_SHIFT)
- #define ATCA_CHIP_MODE_WDG_LONG_SHIFT (2)
- #define ATCA_CHIP_MODE_WDG_LONG_MASK (0x01u << ATCA_CHIP_MODE_WDG_LONG_SHIFT)
- #define ATCA_CHIP_MODE_CLK_DIV_SHIFT (3)
- #define ATCA_CHIP_MODE_CLK_DIV_MASK (0x1Fu << ATCA_CHIP_MODE_CLK_DIV_SHIFT)
- #define ATCA_CHIP_MODE_CLK_DIV(v) (ATCA_CHIP_MODE_CLK_DIV_MASK & (v << ATCA_CHIP_MODE_CLK_DIV_SI
- #define ATCA_SLOT_CONFIG_READKEY_SHIFT (0)
- #define ATCA_SLOT_CONFIG_READKEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_READKEY_SHIFT)
- #define ATCA_SLOT_CONFIG_READKEY(v) (ATCA_SLOT_CONFIG_READKEY_MASK & (v << ATCA_SLOT_CONFIG_READKEY_SHIFT))
- #define ATCA_SLOT_CONFIG_NOMAC_SHIFT (4)
- #define ATCA_SLOT_CONFIG_NOMAC_MASK (0x01u << ATCA_SLOT_CONFIG_NOMAC_SHIFT)
- #define ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT (5)
- #define ATCA_SLOT_CONFIG_LIMITED_USE_MASK (0x01u << ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT)
- #define ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIFT (6)
- #define ATCA_SLOT_CONFIG_ENCRYPTED_READ_MASK (0x01u << ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIF
- #define ATCA_SLOT_CONFIG_IS_SECRET_SHIFT (7)
- #define ATCA_SLOT_CONFIG_IS_SECRET_MASK (0x01u << ATCA_SLOT_CONFIG_IS_SECRET_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT (8)
- #define ATCA_SLOT_CONFIG_WRITE_KEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_KEY(v) (ATCA_SLOT_CONFIG_WRITE_KEY_MASK & (v << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT))
- #define ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT (12)
- #define ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT)

- #define ATCA_SLOT_CONFIG_WRITE_CONFIG(v) (ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK & (v << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT))
- #define ATCA_SLOT_CONFIG_EXT_SIG_SHIFT (0)
- #define ATCA_SLOT_CONFIG_EXT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_EXT_SIG_SHIFT)
- #define ATCA_SLOT_CONFIG_INT_SIG_SHIFT (1)
- #define ATCA_SLOT_CONFIG_INT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_INT_SIG_SHIFT)
- #define ATCA_SLOT_CONFIG_ECDH_SHIFT (2)
- #define ATCA_SLOT_CONFIG_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_ECDH_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT (3)
- #define ATCA_SLOT_CONFIG_WRITE_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT)
- #define ATCA_SLOT_CONFIG_GEN_KEY_SHIFT (8)
- #define ATCA_SLOT_CONFIG_GEN_KEY_MASK (0x01u << ATCA_SLOT_CONFIG_GEN_KEY_SHIFT)
- #define ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT (9)
- #define ATCA_SLOT_CONFIG_PRIV_WRITE_MASK (0x01u << ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT)
- #define ATCA_USE_LOCK_ENABLE_SHIFT (0)
- #define ATCA_USE_LOCK_ENABLE_MASK (0x0Fu << ATCA_USE_LOCK_ENABLE_SHIFT)
- #define ATCA_USE_LOCK_KEY_SHIFT (4)
- #define ATCA_USE_LOCK_KEY_MASK (0x0Fu << ATCA_USE_LOCK_KEY_SHIFT)
- #define ATCA_VOL_KEY_PERM_SLOT_SHIFT (0)
- #define ATCA_VOL_KEY_PERM_SLOT_MASK (0x0Fu << ATCA_VOL_KEY_PERM_SLOT_SHIFT)
- #define ATCA_VOL_KEY_PERM_SLOT(v) (ATCA_VOL_KEY_PERM_SLOT_MASK & (v << ATCA_VOL_KEY_PERM_SLOT
- #define ATCA_VOL_KEY_PERM_EN_SHIFT (7)
- #define ATCA_VOL_KEY_PERM_EN_MASK (0x01u << ATCA_VOL_KEY_PERM_EN_SHIFT)
- #define ATCA_SECURE_BOOT_MODE_SHIFT (0)
- #define ATCA_SECURE_BOOT_MODE_MASK (0x03u << ATCA_SECURE_BOOT_MODE_SHIFT)
- #define ATCA_SECURE_BOOT_MODE(v) (ATCA_SECURE_BOOT_MODE_MASK & (v << ATCA_SECURE_BOOT_MODE_
- #define ATCA_SECURE_BOOT_PERSIST_EN_SHIFT (3)
- #define ATCA_SECURE_BOOT_PERSIST_EN_MASK (0x01u << ATCA_SECURE_BOOT_PERSIST_EN_SHIFT)
- #define ATCA_SECURE_BOOT_RAND_NONCE_SHIFT (4)
- #define ATCA_SECURE_BOOT_RAND_NONCE_MASK (0x01u << ATCA_SECURE_BOOT_RAND_NONCE_SHIFT)
- #define ATCA_SECURE_BOOT_DIGEST_SHIFT (8)
- #define ATCA_SECURE_BOOT_DIGEST_MASK (0x0Fu << ATCA_SECURE_BOOT_DIGEST_SHIFT)
- #define ATCA_SECURE_BOOT_DIGEST(v) (ATCA_SECURE_BOOT_DIGEST_MASK & (v << ATCA_SECURE_BOOT_DIG
- #define ATCA_SECURE_BOOT_PUB_KEY_SHIFT (12)
- #define ATCA_SECURE_BOOT_PUB_KEY_MASK (0x0Fu << ATCA_SECURE_BOOT_PUB_KEY_SHIFT)
- #define ATCA_SECURE_BOOT_PUB_KEY(v) (ATCA_SECURE_BOOT_PUB_KEY_MASK & (v << ATCA_SECURE_BOOT_PUB_KEY_SHIFT))
- #define ATCA_SLOT_LOCKED(v) ((0x01 << v) & 0xFFFFu)
- #define ATCA_CHIP_OPT_POST_EN_SHIFT (0)
- #define ATCA_CHIP_OPT_POST_EN_MASK (0x01u << ATCA_CHIP_OPT_POST_EN_SHIFT)
- #define ATCA_CHIP_OPT_IO_PROT_EN_SHIFT (1)
- #define ATCA_CHIP_OPT_IO_PROT_EN_MASK (0x01u << ATCA_CHIP_OPT_IO_PROT_EN_SHIFT)
- #define ATCA_CHIP_OPT_KDF_AES_EN_SHIFT (2)
- #define ATCA_CHIP_OPT_KDF_AES_EN_MASK (0x01u << ATCA_CHIP_OPT_KDF_AES_EN_SHIFT)
- #define ATCA_CHIP_OPT_ECDH_PROT_SHIFT (8)
- #define ATCA_CHIP_OPT_ECDH_PROT_MASK (0x03u << ATCA_CHIP_OPT_ECDH_PROT_SHIFT)
- #define ATCA_CHIP_OPT_ECDH_PROT(v) (ATCA_CHIP_OPT_ECDH_PROT_MASK & (v << ATCA_CHIP_OPT_ECDH_PR
- #define ATCA_CHIP_OPT_KDF_PROT_SHIFT (10)
- #define ATCA_CHIP_OPT_KDF_PROT_MASK (0x03u << ATCA_CHIP_OPT_KDF_PROT_SHIFT)
- #define ATCA_CHIP_OPT_KDF_PROT(v) (ATCA_CHIP_OPT_KDF_PROT_MASK & (v << ATCA_CHIP_OPT_KDF_PROT_
- #define ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT (12)
- #define ATCA_CHIP_OPT_IO_PROT_KEY_MASK (0x0Fu << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT)
- #define ATCA_CHIP_OPT_IO_PROT_KEY(v) (ATCA_CHIP_OPT_IO_PROT_KEY_MASK & (v << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT))
- #define ATCA_KEY_CONFIG_OFFSET(x) (96UL + (x) ∗ 2)

- #define ATCA_KEY_CONFIG_PRIVATE_SHIFT (0)
- #define ATCA_KEY_CONFIG_PRIVATE_MASK (0x01u << ATCA_KEY_CONFIG_PRIVATE_SHIFT)
- #define ATCA_KEY_CONFIG_PUB_INFO_SHIFT (1)
- #define ATCA_KEY_CONFIG_PUB_INFO_MASK (0x01u << ATCA_KEY_CONFIG_PUB_INFO_SHIFT)
- #define ATCA_KEY_CONFIG_KEY_TYPE_SHIFT (2)
- #define ATCA_KEY_CONFIG_KEY_TYPE_MASK (0x07u << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT)
- #define ATCA_KEY_CONFIG_KEY_TYPE(v) (ATCA_KEY_CONFIG_KEY_TYPE_MASK & (v << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT))
- #define ATCA_KEY_CONFIG_LOCKABLE_SHIFT (5)
- #define ATCA_KEY_CONFIG_LOCKABLE_MASK (0x01u << ATCA_KEY_CONFIG_LOCKABLE_SHIFT)
- #define ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT (6)
- #define ATCA_KEY_CONFIG_REQ_RANDOM_MASK (0x01u << ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT)
- #define ATCA_KEY_CONFIG_REQ_AUTH_SHIFT (7)
- #define ATCA_KEY_CONFIG_REQ_AUTH_MASK (0x01u << ATCA_KEY_CONFIG_REQ_AUTH_SHIFT)
- #define ATCA_KEY_CONFIG_AUTH_KEY_SHIFT (8)
- #define ATCA_KEY_CONFIG_AUTH_KEY_MASK (0x0Fu << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT)
- #define ATCA_KEY_CONFIG_AUTH_KEY(v) (ATCA_KEY_CONFIG_AUTH_KEY_MASK & (v << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT))
- #define ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT (12)
- #define ATCA_KEY_CONFIG_PERSIST_DISABLE_MASK (0x01u << ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT)
- #define ATCA_KEY_CONFIG_RFU_SHIFT (13)
- #define ATCA_KEY_CONFIG_RFU_MASK (0x01u << ATCA_KEY_CONFIG_RFU_SHIFT)
- #define ATCA_KEY_CONFIG_X509_ID_SHIFT (14)
- #define ATCA_KEY_CONFIG_X509_ID_MASK (0x03u << ATCA_KEY_CONFIG_X509_ID_SHIFT)
- #define ATCA_KEY_CONFIG_X509_ID(v) (ATCA_KEY_CONFIG_X509_ID_MASK & (v << ATCA_KEY_CONFIG_X509_ID_

Typedefs

- typedef struct _atsha204a_config atsha204a_config_t
- typedef struct _atecc508a_config atecc508a_config_t
- typedef struct _atecc608_config atecc608_config_t
- typedef struct atca_device * ATCADevice

Enumerations

- enum ATCADeviceState { ATCA_DEVICE_STATE_UNKNOWN = 0, ATCA_DEVICE_STATE_SLEEP, ATCA_DEVICE_STATE_IDLE, ATCA_DEVICE_STATE_ACTIVE }

  ATCADeviceState says about device state.
- enum ATCADeviceType {
  ATSHA204A = 0, ATECC108A = 1, ATECC508A = 2, ATECC608A = 3,
  ATECC608B = 3, ATECC608 = 3, ATSHA206A = 4, ECC204 = 5,
  TA100 = 0x10, ATCA_DEV_UNKNOWN = 0x20 }

  The supported Device type in Cryptoauthlib library.

Functions

- ATCADevice newATCADevice (ATCAIfaceCfg *cfg)

  constructor for a Microchip CryptoAuth device
- void deleteATCADevice (ATCADevice *ca_dev)

  destructor for a device NULLs reference after object is freed
- ATCA_STATUS initATCADevice (ATCAIfaceCfg *cfg, ATCADevice ca_dev)

  Initializer for an Microchip CryptoAuth device.
- ATCAIface atGetIFace (ATCADevice dev)

  returns a reference to the ATCAIface interface object for the device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

  Release any resources associated with the device.

8.3.1 Detailed Description

ATCADevice object - composite of command and interface objects.

8.3.2 Macro Definition Documentation

8.3.2.1 ATCA_AES_ENABLE_EN_MASK

```
#define ATCA_AES_ENABLE_EN_MASK (0x01u << ATCA_AES_ENABLE_EN_SHIFT)
```

8.3.2.2 ATCA_AES_ENABLE_EN_SHIFT

```
#define ATCA_AES_ENABLE_EN_SHIFT (0)
```

8.3.2.3 ATCA_CHIP_MODE_CLK_DIV

```
#define ATCA_CHIP_MODE_CLK_DIV(
              v ) (ATCA_CHIP_MODE_CLK_DIV_MASK & (v << ATCA_CHIP_MODE_CLK_DIV_SHIFT))
```

8.3.2.4 ATCA_CHIP_MODE_CLK_DIV_MASK

```
#define ATCA_CHIP_MODE_CLK_DIV_MASK (0x1Fu << ATCA_CHIP_MODE_CLK_DIV_SHIFT)
```

8.3.2.5 ATCA_CHIP_MODE_CLK_DIV_SHIFT

```
#define ATCA_CHIP_MODE_CLK_DIV_SHIFT (3)
```

8.3.2.6 ATCA_CHIP_MODE_I2C_EXTRA_MASK

```
#define ATCA_CHIP_MODE_I2C_EXTRA_MASK (0x01u << ATCA_CHIP_MODE_I2C_EXTRA_SHIFT)
```

8.3.2.7 ATCA_CHIP_MODE_I2C_EXTRA_SHIFT

```
#define ATCA_CHIP_MODE_I2C_EXTRA_SHIFT (0)
```

8.3.2.8 ATCA_CHIP_MODE_TTL_EN_MASK

```
#define ATCA_CHIP_MODE_TTL_EN_MASK (0x01u << ATCA_CHIP_MODE_TTL_EN_SHIFT)
```

8.3.2.9 ATCA_CHIP_MODE_TTL_EN_SHIFT

```
#define ATCA_CHIP_MODE_TTL_EN_SHIFT (1)
```

8.3.2.10 ATCA_CHIP_MODE_WDG_LONG_MASK

```
#define ATCA_CHIP_MODE_WDG_LONG_MASK (0x01u << ATCA_CHIP_MODE_WDG_LONG_SHIFT)
```

8.3.2.11 ATCA_CHIP_MODE_WDG_LONG_SHIFT

```
#define ATCA_CHIP_MODE_WDG_LONG_SHIFT (2)
```

8.3.2.12 ATCA_CHIP_OPT_ECDH_PROT

```
#define ATCA_CHIP_OPT_ECDH_PROT(
              v ) (ATCA_CHIP_OPT_ECDH_PROT_MASK & (v << ATCA_CHIP_OPT_ECDH_PROT_SHIFT))
```

8.3.2.13 ATCA_CHIP_OPT_ECDH_PROT_MASK

```
#define ATCA_CHIP_OPT_ECDH_PROT_MASK (0x03u << ATCA_CHIP_OPT_ECDH_PROT_SHIFT)
```

8.3.2.14 ATCA_CHIP_OPT_ECDH_PROT_SHIFT

```
#define ATCA_CHIP_OPT_ECDH_PROT_SHIFT (8)
```

8.3.2.15 ATCA_CHIP_OPT_IO_PROT_EN_MASK

```
#define ATCA_CHIP_OPT_IO_PROT_EN_MASK (0x01u << ATCA_CHIP_OPT_IO_PROT_EN_SHIFT)
```

8.3.2.16 ATCA_CHIP_OPT_IO_PROT_EN_SHIFT

```
#define ATCA_CHIP_OPT_IO_PROT_EN_SHIFT (1)
```

8.3.2.17 ATCA_CHIP_OPT_IO_PROT_KEY

```
#define ATCA_CHIP_OPT_IO_PROT_KEY(
              v ) (ATCA_CHIP_OPT_IO_PROT_KEY_MASK & (v << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT))
```

8.3.2.18 ATCA_CHIP_OPT_IO_PROT_KEY_MASK

```
#define ATCA_CHIP_OPT_IO_PROT_KEY_MASK (0x0Fu << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT)
```

8.3.2.19 ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT

```
#define ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT (12)
```

8.3.2.20 ATCA_CHIP_OPT_KDF_AES_EN_MASK

```
#define ATCA_CHIP_OPT_KDF_AES_EN_MASK (0x01u << ATCA_CHIP_OPT_KDF_AES_EN_SHIFT)
```

8.3.2.21 ATCA_CHIP_OPT_KDF_AES_EN_SHIFT

```
#define ATCA_CHIP_OPT_KDF_AES_EN_SHIFT (2)
```

8.3.2.22 ATCA_CHIP_OPT_KDF_PROT

```
#define ATCA_CHIP_OPT_KDF_PROT(
            v ) (ATCA_CHIP_OPT_KDF_PROT_MASK & (v << ATCA_CHIP_OPT_KDF_PROT_SHIFT))
```

8.3.2.23 ATCA_CHIP_OPT_KDF_PROT_MASK

```
#define ATCA_CHIP_OPT_KDF_PROT_MASK (0x03u << ATCA_CHIP_OPT_KDF_PROT_SHIFT)
```

8.3.2.24 ATCA_CHIP_OPT_KDF_PROT_SHIFT

```
#define ATCA_CHIP_OPT_KDF_PROT_SHIFT (10)
```

8.3.2.25 ATCA_CHIP_OPT_POST_EN_MASK

```
#define ATCA_CHIP_OPT_POST_EN_MASK (0x01u << ATCA_CHIP_OPT_POST_EN_SHIFT)
```

8.3.2.26 ATCA_CHIP_OPT_POST_EN_SHIFT

```
#define ATCA_CHIP_OPT_POST_EN_SHIFT (0)
```

8.3.2.27 ATCA_COUNTER_MATCH_EN_MASK

```
#define ATCA_COUNTER_MATCH_EN_MASK (0x01u << ATCA_COUNTER_MATCH_EN_SHIFT)
```

8.3.2.28 ATCA_COUNTER_MATCH_EN_SHIFT

```
#define ATCA_COUNTER_MATCH_EN_SHIFT (0)
```

8.3.2.29 ATCA_COUNTER_MATCH_KEY

```
#define ATCA_COUNTER_MATCH_KEY(
              v ) (ATCA_COUNTER_MATCH_KEY_MASK & (v << ATCA_COUNTER_MATCH_KEY_SHIFT))
```

8.3.2.30 ATCA_COUNTER_MATCH_KEY_MASK

```
#define ATCA_COUNTER_MATCH_KEY_MASK (0x0Fu << ATCA_COUNTER_MATCH_KEY_SHIFT)
```

8.3.2.31 ATCA_COUNTER_MATCH_KEY_SHIFT

```
#define ATCA_COUNTER_MATCH_KEY_SHIFT (4)
```

8.3.2.32 ATCA_I2C_ENABLE_EN_MASK

```
#define ATCA_I2C_ENABLE_EN_MASK (0x01u << ATCA_I2C_ENABLE_EN_SHIFT)
```

8.3.2.33 ATCA_I2C_ENABLE_EN_SHIFT

```
#define ATCA_I2C_ENABLE_EN_SHIFT (0)
```

8.3.2.34 ATCA_KEY_CONFIG_AUTH_KEY

```
#define ATCA_KEY_CONFIG_AUTH_KEY(
              v ) (ATCA_KEY_CONFIG_AUTH_KEY_MASK & (v << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT))
```

8.3.2.35 ATCA_KEY_CONFIG_AUTH_KEY_MASK

```
#define ATCA_KEY_CONFIG_AUTH_KEY_MASK (0x0Fu << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT)
```

8.3.2.36 ATCA_KEY_CONFIG_AUTH_KEY_SHIFT

```
#define ATCA_KEY_CONFIG_AUTH_KEY_SHIFT (8)
```

8.3.2.37 ATCA_KEY_CONFIG_KEY_TYPE

```
#define ATCA_KEY_CONFIG_KEY_TYPE(
              v ) (ATCA_KEY_CONFIG_KEY_TYPE_MASK & (v << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT))
```

8.3.2.38 ATCA_KEY_CONFIG_KEY_TYPE_MASK

```
#define ATCA_KEY_CONFIG_KEY_TYPE_MASK (0x07u << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT)
```

8.3.2.39 ATCA_KEY_CONFIG_KEY_TYPE_SHIFT

```
#define ATCA_KEY_CONFIG_KEY_TYPE_SHIFT (2)
```

8.3.2.40 ATCA_KEY_CONFIG_LOCKABLE_MASK

```
#define ATCA_KEY_CONFIG_LOCKABLE_MASK (0x01u << ATCA_KEY_CONFIG_LOCKABLE_SHIFT)
```

8.3.2.41 ATCA_KEY_CONFIG_LOCKABLE_SHIFT

```
#define ATCA_KEY_CONFIG_LOCKABLE_SHIFT (5)
```

8.3.2.42 ATCA_KEY_CONFIG_OFFSET

```
#define ATCA_KEY_CONFIG_OFFSET(
              x ) (96UL + (x) * 2)
```

8.3.2.43 ATCA_KEY_CONFIG_PERSIST_DISABLE_MASK

```
#define ATCA_KEY_CONFIG_PERSIST_DISABLE_MASK (0x01u << ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT)
```

8.3.2.44 ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT

```
#define ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT (12)
```

8.3.2.45 ATCA_KEY_CONFIG_PRIVATE_MASK

```
#define ATCA_KEY_CONFIG_PRIVATE_MASK (0x01u << ATCA_KEY_CONFIG_PRIVATE_SHIFT)
```

8.3.2.46 ATCA_KEY_CONFIG_PRIVATE_SHIFT

```
#define ATCA_KEY_CONFIG_PRIVATE_SHIFT (0)
```

8.3.2.47 ATCA_KEY_CONFIG_PUB_INFO_MASK

```
#define ATCA_KEY_CONFIG_PUB_INFO_MASK (0x01u << ATCA_KEY_CONFIG_PUB_INFO_SHIFT)
```

8.3.2.48 ATCA_KEY_CONFIG_PUB_INFO_SHIFT

```
#define ATCA_KEY_CONFIG_PUB_INFO_SHIFT (1)
```

8.3.2.49 ATCA_KEY_CONFIG_REQ_AUTH_MASK

```
#define ATCA_KEY_CONFIG_REQ_AUTH_MASK (0x01u << ATCA_KEY_CONFIG_REQ_AUTH_SHIFT)
```

8.3.2.50 ATCA_KEY_CONFIG_REQ_AUTH_SHIFT

```
#define ATCA_KEY_CONFIG_REQ_AUTH_SHIFT (7)
```

8.3.2.51 ATCA_KEY_CONFIG_REQ_RANDOM_MASK

```
#define ATCA_KEY_CONFIG_REQ_RANDOM_MASK (0x01u << ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT)
```

8.3.2.52 ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT

```
#define ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT (6)
```

8.3.2.53 ATCA_KEY_CONFIG_RFU_MASK

```
#define ATCA_KEY_CONFIG_RFU_MASK (0x01u << ATCA_KEY_CONFIG_RFU_SHIFT)
```

8.3.2.54 ATCA_KEY_CONFIG_RFU_SHIFT

```
#define ATCA_KEY_CONFIG_RFU_SHIFT (13)
```

8.3.2.55 ATCA_KEY_CONFIG_X509_ID

```
#define ATCA_KEY_CONFIG_X509_ID(
              v ) (ATCA_KEY_CONFIG_X509_ID_MASK & (v << ATCA_KEY_CONFIG_X509_ID_SHIFT))
```

8.3.2.56 ATCA_KEY_CONFIG_X509_ID_MASK

```
#define ATCA_KEY_CONFIG_X509_ID_MASK (0x03u << ATCA_KEY_CONFIG_X509_ID_SHIFT)
```

8.3.2.57 ATCA_KEY_CONFIG_X509_ID_SHIFT

```
#define ATCA_KEY_CONFIG_X509_ID_SHIFT (14)
```

8.3.2.58 ATCA_PACKED

```
#define ATCA_PACKED
```

8.3.2.59 ATCA_SECURE_BOOT_DIGEST

```
#define ATCA_SECURE_BOOT_DIGEST(
              v ) (ATCA_SECURE_BOOT_DIGEST_MASK & (v << ATCA_SECURE_BOOT_DIGEST_SHIFT))
```

8.3.2.60 ATCA_SECURE_BOOT_DIGEST_MASK

```
#define ATCA_SECURE_BOOT_DIGEST_MASK (0x0Fu << ATCA_SECURE_BOOT_DIGEST_SHIFT)
```

8.3.2.61 ATCA_SECURE_BOOT_DIGEST_SHIFT

```
#define ATCA_SECURE_BOOT_DIGEST_SHIFT (8)
```

8.3.2.62 ATCA_SECURE_BOOT_MODE

```
#define ATCA_SECURE_BOOT_MODE(
              v ) (ATCA_SECURE_BOOT_MODE_MASK & (v << ATCA_SECURE_BOOT_MODE_SHIFT))
```

8.3.2.63 ATCA_SECURE_BOOT_MODE_MASK

```
#define ATCA_SECURE_BOOT_MODE_MASK (0x03u << ATCA_SECURE_BOOT_MODE_SHIFT)
```

8.3.2.64 ATCA_SECURE_BOOT_MODE_SHIFT

```
#define ATCA_SECURE_BOOT_MODE_SHIFT (0)
```

8.3.2.65 ATCA_SECURE_BOOT_PERSIST_EN_MASK

```
#define ATCA_SECURE_BOOT_PERSIST_EN_MASK (0x01u << ATCA_SECURE_BOOT_PERSIST_EN_SHIFT)
```

8.3.2.66 ATCA_SECURE_BOOT_PERSIST_EN_SHIFT

```
#define ATCA_SECURE_BOOT_PERSIST_EN_SHIFT (3)
```

8.3.2.67 ATCA_SECURE_BOOT_PUB_KEY

```
#define ATCA_SECURE_BOOT_PUB_KEY(
              v ) (ATCA_SECURE_BOOT_PUB_KEY_MASK & (v << ATCA_SECURE_BOOT_PUB_KEY_SHIFT))
```

8.3.2.68 ATCA_SECURE_BOOT_PUB_KEY_MASK

```
#define ATCA_SECURE_BOOT_PUB_KEY_MASK (0x0Fu << ATCA_SECURE_BOOT_PUB_KEY_SHIFT)
```

8.3.2.69 ATCA_SECURE_BOOT_PUB_KEY_SHIFT

```
#define ATCA_SECURE_BOOT_PUB_KEY_SHIFT (12)
```

8.3.2.70 ATCA_SECURE_BOOT_RAND_NONCE_MASK

```
#define ATCA_SECURE_BOOT_RAND_NONCE_MASK (0x01u << ATCA_SECURE_BOOT_RAND_NONCE_SHIFT)
```

8.3.2.71 ATCA_SECURE_BOOT_RAND_NONCE_SHIFT

```
#define ATCA_SECURE_BOOT_RAND_NONCE_SHIFT (4)
```

8.3.2.72 ATCA_SLOT_CONFIG_ECDH_MASK

```
#define ATCA_SLOT_CONFIG_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_ECDH_SHIFT)
```

8.3.2.73 ATCA_SLOT_CONFIG_ECDH_SHIFT

```
#define ATCA_SLOT_CONFIG_ECDH_SHIFT (2)
```

8.3.2.74 ATCA_SLOT_CONFIG_ENCRYPTED_READ_MASK

```
#define ATCA_SLOT_CONFIG_ENCRYPTED_READ_MASK (0x01u << ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIFT)
```

8.3.2.75 ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIFT

```
#define ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIFT (6)
```

8.3.2.76 ATCA_SLOT_CONFIG_EXT_SIG_MASK

```
#define ATCA_SLOT_CONFIG_EXT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_EXT_SIG_SHIFT)
```

8.3.2.77 ATCA_SLOT_CONFIG_EXT_SIG_SHIFT

```
#define ATCA_SLOT_CONFIG_EXT_SIG_SHIFT (0)
```

8.3.2.78 ATCA_SLOT_CONFIG_GEN_KEY_MASK

```
#define ATCA_SLOT_CONFIG_GEN_KEY_MASK (0x01u << ATCA_SLOT_CONFIG_GEN_KEY_SHIFT)
```

8.3.2.79 ATCA_SLOT_CONFIG_GEN_KEY_SHIFT

```
#define ATCA_SLOT_CONFIG_GEN_KEY_SHIFT (8)
```

8.3.2.80 ATCA_SLOT_CONFIG_INT_SIG_MASK

```
#define ATCA_SLOT_CONFIG_INT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_INT_SIG_SHIFT)
```

8.3.2.81 ATCA_SLOT_CONFIG_INT_SIG_SHIFT

```
#define ATCA_SLOT_CONFIG_INT_SIG_SHIFT (1)
```

8.3.2.82 ATCA_SLOT_CONFIG_IS_SECRET_MASK

```
#define ATCA_SLOT_CONFIG_IS_SECRET_MASK (0x01u << ATCA_SLOT_CONFIG_IS_SECRET_SHIFT)
```

8.3.2.83 ATCA_SLOT_CONFIG_IS_SECRET_SHIFT

```
#define ATCA_SLOT_CONFIG_IS_SECRET_SHIFT (7)
```

8.3.2.84 ATCA_SLOT_CONFIG_LIMITED_USE_MASK

```
#define ATCA_SLOT_CONFIG_LIMITED_USE_MASK (0x01u << ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT)
```

8.3.2.85 ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT

```
#define ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT (5)
```

8.3.2.86 ATCA_SLOT_CONFIG_NOMAC_MASK

```
#define ATCA_SLOT_CONFIG_NOMAC_MASK (0x01u << ATCA_SLOT_CONFIG_NOMAC_SHIFT)
```

8.3.2.87 ATCA_SLOT_CONFIG_NOMAC_SHIFT

```
#define ATCA_SLOT_CONFIG_NOMAC_SHIFT (4)
```

8.3.2.88 ATCA_SLOT_CONFIG_PRIV_WRITE_MASK

```
#define ATCA_SLOT_CONFIG_PRIV_WRITE_MASK (0x01u << ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT)
```

8.3.2.89 ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT

```
#define ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT (9)
```

8.3.2.90 ATCA_SLOT_CONFIG_READKEY

```
#define ATCA_SLOT_CONFIG_READKEY(
              v ) (ATCA_SLOT_CONFIG_READKEY_MASK & (v << ATCA_SLOT_CONFIG_READKEY_SHIFT))
```

8.3.2.91 ATCA_SLOT_CONFIG_READKEY_MASK

```
#define ATCA_SLOT_CONFIG_READKEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_READKEY_SHIFT)
```

8.3.2.92 ATCA_SLOT_CONFIG_READKEY_SHIFT

```
#define ATCA_SLOT_CONFIG_READKEY_SHIFT (0)
```

8.3.2.93 ATCA_SLOT_CONFIG_WRITE_CONFIG

```
#define ATCA_SLOT_CONFIG_WRITE_CONFIG(
            v ) (ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK & (v << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT))
```

8.3.2.94 ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK

```
#define ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT)
```

8.3.2.95 ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT

```
#define ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT (12)
```

8.3.2.96 ATCA_SLOT_CONFIG_WRITE_ECDH_MASK

```
#define ATCA_SLOT_CONFIG_WRITE_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT)
```

8.3.2.97 ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT

```
#define ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT (3)
```

8.3.2.98 ATCA_SLOT_CONFIG_WRITE_KEY

```
#define ATCA_SLOT_CONFIG_WRITE_KEY(
            v ) (ATCA_SLOT_CONFIG_WRITE_KEY_MASK & (v << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT))
```

8.3.2.99 ATCA_SLOT_CONFIG_WRITE_KEY_MASK

```
#define ATCA_SLOT_CONFIG_WRITE_KEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT)
```

8.3.2.100 ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT

```
#define ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT (8)
```

8.3.2.101 ATCA_SLOT_LOCKED

```
#define ATCA_SLOT_LOCKED(
              v ) ((0x01 << v) & 0xFFFFu)
```

8.3.2.102 ATCA_USE_LOCK_ENABLE_MASK

```
#define ATCA_USE_LOCK_ENABLE_MASK (0x0Fu << ATCA_USE_LOCK_ENABLE_SHIFT)
```

8.3.2.103 ATCA_USE_LOCK_ENABLE_SHIFT

```
#define ATCA_USE_LOCK_ENABLE_SHIFT (0)
```

8.3.2.104 ATCA_USE_LOCK_KEY_MASK

```
#define ATCA_USE_LOCK_KEY_MASK (0x0Fu << ATCA_USE_LOCK_KEY_SHIFT)
```

8.3.2.105 ATCA_USE_LOCK_KEY_SHIFT

```
#define ATCA_USE_LOCK_KEY_SHIFT (4)
```

8.3.2.106 ATCA_VOL_KEY_PERM_EN_MASK

```
#define ATCA_VOL_KEY_PERM_EN_MASK (0x01u << ATCA_VOL_KEY_PERM_EN_SHIFT)
```

8.3.2.107 ATCA_VOL_KEY_PERM_EN_SHIFT

```
#define ATCA_VOL_KEY_PERM_EN_SHIFT (7)
```

8.3.2.108 ATCA_VOL_KEY_PERM_SLOT

```
#define ATCA_VOL_KEY_PERM_SLOT(
              v ) (ATCA_VOL_KEY_PERM_SLOT_MASK & (v << ATCA_VOL_KEY_PERM_SLOT_SHIFT))
```

8.3.2.109 ATCA_VOL_KEY_PERM_SLOT_MASK

```
#define ATCA_VOL_KEY_PERM_SLOT_MASK (0x0Fu << ATCA_VOL_KEY_PERM_SLOT_SHIFT)
```

8.3.2.110 ATCA_VOL_KEY_PERM_SLOT_SHIFT

```
#define ATCA_VOL_KEY_PERM_SLOT_SHIFT (0)
```

8.3.3 Typedef Documentation

8.3.3.1 ATCADevice

```
typedef struct atca_device* ATCADevice
```

8.3.3.2 atecc508a_config_t

```
typedef struct _atecc508a_config atecc508a_config_t
```

8.3.3.3 atecc608_config_t

```
typedef struct _atecc608_config atecc608_config_t
```

8.3.3.4 atsha204a_config_t

```
typedef struct _atsha204a_config atsha204a_config_t
```

8.3.4 Enumeration Type Documentation

8.3.4.1 ATCADeviceState

```
enum ATCADeviceState
```

ATCADeviceState says about device state.

Enumerator

ATCA_DEVICE_STATE_UNKNOWN	
ATCA_DEVICE_STATE_SLEEP	
ATCA_DEVICE_STATE_IDLE	
ATCA_DEVICE_STATE_ACTIVE	

8.3.4.2 ATCADeviceType

```
enum ATCADeviceType
```

The supported Device type in Cryptoauthlib library.

Enumerator

ATSHA204A	
ATECC108A	
ATECC508A	
ATECC608A	
ATECC608B	
ATECC608	
ATSHA206A	
ECC204	
TA100	
ATCA_DEV_UNKNOWN	

8.3.5 Function Documentation

8.3.5.1 atGetIFace()

```
ATCAIface atGetIFace (
            ATCADevice dev )
```

returns a reference to the ATCAIface interface object for the device

Parameters

in	*dev*	reference to a device

Returns

reference to the ATCAIface object for the device

8.3.5.2 deleteATCADevice()

```
void deleteATCADevice (
            ATCADevice * ca_dev )
```

destructor for a device NULLs reference after object is freed

Parameters

in	*ca_dev*	pointer to a reference to a device

8.3.5.3 initATCADevice()

```
ATCA_STATUS initATCADevice (
            ATCAIfaceCfg * cfg,
            ATCADevice ca_dev )
```

Initializer for an Microchip CryptoAuth device.

Parameters

in	*cfg*	pointer to an interface configuration object
in,out	*ca_dev*	As input, pre-allocated structure to be initialized. mCommands and mIface members should point to existing structures to be initialized.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.3.5.4 newATCADevice()

```
ATCADevice newATCADevice (
            ATCAIfaceCfg * cfg )
```

constructor for a Microchip CryptoAuth device

Parameters

in	*cfg*	Interface configuration object

Returns

Reference to a new ATCADevice on success. NULL on failure.

8.3.5.5 releaseATCADevice()

```
ATCA_STATUS releaseATCADevice (
            ATCADevice ca_dev )
```

Release any resources associated with the device.

Parameters

in	*ca_dev*	Device to release

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4 ATCAIface (atca_)

Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.

Data Structures

- struct ATCAIfaceCfg
- struct ATCAHAL_t

  HAL Driver Structure.
- struct atca_iface

  atca_iface is the context structure for a configured interface

Typedefs

- typedef struct atca_iface ∗ ATCAIface
- typedef struct atca_iface atca_iface_t

  atca_iface is the context structure for a configured interface

Enumerations

- enum ATCAIfaceType {
  ATCA_I2C_IFACE = 0, ATCA_SWI_IFACE = 1, ATCA_UART_IFACE = 2, ATCA_SPI_IFACE = 3,
  ATCA_HID_IFACE = 4, ATCA_KIT_IFACE = 5, ATCA_CUSTOM_IFACE = 6, ATCA_I2C_GPIO_IFACE = 7,
  ATCA_SWI_GPIO_IFACE = 8, ATCA_SPI_GPIO_IFACE = 9, ATCA_UNKNOWN_IFACE = 0xFE }
- enum ATCAKitType {
  ATCA_KIT_AUTO_IFACE, ATCA_KIT_I2C_IFACE, ATCA_KIT_SWI_IFACE, ATCA_KIT_SPI_IFACE,
  ATCA_KIT_UNKNOWN_IFACE }

Functions

- ATCA_STATUS initATCAIface (ATCAIfaceCfg ∗cfg, ATCAIface ca_iface)

  Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg ∗cfg)

  Constructor for ATCAIface objects.
- ATCA_STATUS atinit (ATCAIface ca_iface)

  Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t address, uint8_t ∗txdata, int txlength)

  Sends the data to the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atcontrol (ATCAIface ca_iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations with the underlying hal driver.
- ATCA_STATUS atwake (ATCAIface ca_iface)

  Wakes up the device by calling intermediate HAL wrapper function. The atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

Puts the device into idle state by calling intermediate HAL wrapper function. The atcab_idle() function should be used instead.

- ATCA_STATUS atsleep (ATCAIface ca_iface)

  Puts the device into sleep state by calling intermediate HAL wrapper function. The atcab_sleep() function should be used instead.
- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

  Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

  Returns the HAL data pointer for the device.
- bool atca_iface_is_kit (ATCAIface ca_iface)

  Check if the given interface is configured as a "kit protocol" one where transactions are atomic.
- int atca_iface_get_retries (ATCAIface ca_iface)

  Retrive the number of retries for a configured interface.
- uint16_t atca_iface_get_wake_delay (ATCAIface ca_iface)

  Retrive the wake/retry delay for a configured interface/device.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

  Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface *ca_iface)

  Instruct the HAL driver to release any resources associated with this interface, then delete the object.

8.4.1 Detailed Description

Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.

8.4.2 Typedef Documentation

8.4.2.1 atca_iface_t

```
typedef struct atca_iface atca_iface_t
```

atca_iface is the context structure for a configured interface

8.4.2.2 ATCAIface

```
typedef struct atca_iface* ATCAIface
```

8.4.3 Enumeration Type Documentation

8.4.3.1 ATCAIfaceType

```
enum ATCAIfaceType
```

Enumerator

ATCA_I2C_IFACE	Native I2C Driver
ATCA_SWI_IFACE	SWI or 1-Wire over UART/USART
ATCA_UART_IFACE	Kit v1 over UART/USART
ATCA_SPI_IFACE	Native SPI Driver
ATCA_HID_IFACE	Kit v1 over HID
ATCA_KIT_IFACE	Kit v2 (Binary/Bridging)
ATCA_CUSTOM_IFACE	Custom HAL functions provided during interface init
ATCA_I2C_GPIO_IFACE	I2C "Bitbang" Driver
ATCA_SWI_GPIO_IFACE	SWI or 1-Wire using a GPIO
ATCA_SPI_GPIO_IFACE	SWI or 1-Wire using a GPIO
ATCA_UNKNOWN_IFACE	

8.4.3.2 ATCAKitType

```
enum ATCAKitType
```

Enumerator

ATCA_KIT_AUTO_IFACE	
ATCA_KIT_I2C_IFACE	
ATCA_KIT_SWI_IFACE	
ATCA_KIT_SPI_IFACE	
ATCA_KIT_UNKNOWN_IFACE	

8.4.4 Function Documentation

8.4.4.1 atca_iface_get_retries()

```
int atca_iface_get_retries (
            ATCAIface ca_iface )
```

Retrive the number of retries for a configured interface.

8.4.4.2 atca_iface_get_wake_delay()

```
uint16_t atca_iface_get_wake_delay (
            ATCAIface ca_iface )
```

Retrive the wake/retry delay for a configured interface/device.

8.4.4.3 atca_iface_is_kit()

```
bool atca_iface_is_kit (
            ATCAIface ca_iface )
```

Check if the given interface is configured as a "kit protocol" one where transactions are atomic.

Returns

true if the interface is considered a kit

8.4.4.4 atcontrol()

```
ATCA_STATUS atcontrol (
            ATCAIface ca_iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations with the underlying hal driver.

Parameters

in	*ca_iface*	Device to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.5 atgetifacecfg()

```
ATCAIfaceCfg * atgetifacecfg (
            ATCAIface ca_iface )
```

Returns the logical interface configuration for the device.

Parameters

in	*ca_iface*	Device interface.

Returns

Logical interface configuration.

8.4.4.6 atgetifacehaldat()

```
void * atgetifacehaldat (
            ATCAIface ca_iface )
```

Returns the HAL data pointer for the device.

Parameters

in	*ca_iface*	Device interface.

Returns

HAL data pointer.

8.4.4.7 atidle()

```
ATCA_STATUS atidle (
            ATCAIface ca_iface )
```

Puts the device into idle state by calling intermediate HAL wrapper function. The atcab_idle() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.8 atinit()

```
ATCA_STATUS atinit (
            ATCAIface ca_iface )
```

Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.9 atreceive()

```
ATCA_STATUS atreceive (
            ATCAIface ca_iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

Receives data from the device by calling intermediate HAL wrapper function.

Parameters

in	*ca_iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.10 atsend()

```
ATCA_STATUS atsend (
            ATCAIface ca_iface,
            uint8_t address,
            uint8_t * txdata,
            int txlength )
```

Sends the data to the device by calling intermediate HAL wrapper function.

Parameters

in	*ca_iface*	Device to interact with.
in	*word_address*	device transaction type
in	*txdata*	Data to be transmitted to the device.
in	*txlength*	Number of bytes to be transmitted to the device.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.11 atsleep()

```
ATCA_STATUS atsleep (
            ATCAIface ca_iface )
```

Puts the device into sleep state by calling intermediate HAL wrapper function. The atcab_sleep() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.12 atwake()

```
ATCA_STATUS atwake (
            ATCAIface ca_iface )
```

Wakes up the device by calling intermediate HAL wrapper function. The atcab_wakeup() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.13 deleteATCAIface()

```
void deleteATCAIface (
            ATCAIface * ca_iface )
```

Instruct the HAL driver to release any resources associated with this interface, then delete the object.

Parameters

in	*ca_iface*	Device interface.

8.4.4.14 initATCAIface()

```
ATCA_STATUS initATCAIface (
            ATCAIfaceCfg * cfg,
            ATCAIface ca_iface )
```

Initializer for ATCAIface objects.

Parameters

in	*cfg*	Logical configuration for the interface
in	*ca_iface*	Interface structure to initialize.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.4.4.15 newATCAIface()

```
ATCAIface newATCAIface (
            ATCAIfaceCfg * cfg )
```

Constructor for ATCAIface objects.

Parameters

in	*cfg*	Logical configuration for the interface

Returns

New interface instance on success. NULL on failure.

8.4.4.16 releaseATCAIface()

```
ATCA_STATUS releaseATCAIface (
            ATCAIface ca_iface )
```

Instruct the HAL driver to release any resources associated with this interface.

Parameters

in	*ca_iface*	Device interface.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.5 Certificate manipulation methods (atcacert_)

These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.

Data Structures

- struct atcacert_tm_utc_s
- struct atcacert_device_loc_s
- struct atcacert_cert_loc_s
- struct atcacert_cert_element_s
- struct atcacert_def_s
- struct atcacert_build_state_s

Macros

- #define FALSE (0)
- #define TRUE (1)
- #define ATCACERT_E_SUCCESS 0

  Operation completed successfully.
- #define ATCACERT_E_ERROR 1

  General error.
- #define ATCACERT_E_BAD_PARAMS 2

  Invalid/bad parameter passed to function.
- #define ATCACERT_E_BUFFER_TOO_SMALL 3

  Supplied buffer for output is too small to hold the result.
- #define ATCACERT_E_DECODING_ERROR 4

  Data being decoded/parsed has an invalid format.
- #define ATCACERT_E_INVALID_DATE 5

  Date is invalid.
- #define ATCACERT_E_UNIMPLEMENTED 6

  Function is unimplemented for the current configuration.
- #define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7

  A certificate element size was not what was expected.
- #define ATCACERT_E_ELEM_MISSING 8

  The certificate element isn't defined for the certificate definition.
- #define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9

  Certificate element is out of bounds for the given certificate.
- #define ATCACERT_E_BAD_CERT 10

  Certificate structure is bad in some way.
- #define ATCACERT_E_WRONG_CERT_DEF 11
- #define ATCACERT_E_VERIFY_FAILED 12

  Certificate or challenge/response verification failed.
- #define ATCACERT_E_INVALID_TRANSFORM 13

  Invalid transform passed to function.
- #define DATEFMT_ISO8601_SEP 0

  ISO8601 full date YYYY-MM-DDThh:mm:ssZ.
- #define DATEFMT_RFC5280_UTC 1

  RFC 5280 (X.509) 4.1.2.5.1 UTCTime format YYMMDDhhmmssZ.
- #define DATEFMT_POSIX_UINT32_BE 2

POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, big endian.

- #define DATEFMT_POSIX_UINT32_LE 3

  POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, little endian.
- #define DATEFMT_RFC5280_GEN 4

  RFC 5280 (X.509) 4.1.2.5.2 GeneralizedTime format YYYYMMDDhhmmssZ.
- #define DATEFMT_ISO8601_SEP_SIZE (20)
- #define DATEFMT_RFC5280_UTC_SIZE (13)
- #define DATEFMT_POSIX_UINT32_BE_SIZE (4)
- #define DATEFMT_POSIX_UINT32_LE_SIZE (4)
- #define DATEFMT_RFC5280_GEN_SIZE (15)
- #define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
- #define ATCACERT_DATE_FORMAT_SIZES_COUNT 5
- #define ATCA_PACKED

Typedefs

- typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
- typedef uint8_t atcacert_date_format_t
- typedef enum atcacert_cert_type_e atcacert_cert_type_t
- typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
- typedef enum atcacert_device_zone_e atcacert_device_zone_t
- typedef enum atcacert_transform_e atcacert_transform_t

  How to transform the data from the device to the certificate.
- typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
- typedef struct atcacert_device_loc_s atcacert_device_loc_t
- typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
- typedef struct atcacert_cert_element_s atcacert_cert_element_t
- typedef struct atcacert_def_s atcacert_def_t
- typedef struct atcacert_build_state_s atcacert_build_state_t

Enumerations

- enum atcacert_cert_type_e { CERTTYPE_X509, CERTTYPE_CUSTOM }
- enum atcacert_cert_sn_src_e {
  SNSRC_STORED = 0x0, SNSRC_STORED_DYNAMIC = 0x7, SNSRC_DEVICE_SN = 0x8, SNSRC_SIGNER_ID = 0x9,
  SNSRC_PUB_KEY_HASH = 0xA, SNSRC_DEVICE_SN_HASH = 0xB, SNSRC_PUB_KEY_HASH_POS = 0xC, SNSRC_DEVICE_SN_HASH_POS = 0xD,
  SNSRC_PUB_KEY_HASH_RAW = 0xE, SNSRC_DEVICE_SN_HASH_RAW = 0xF }
- enum atcacert_device_zone_e { DEVZONE_CONFIG = 0x00, DEVZONE_OTP = 0x01, DEVZONE_DATA = 0x02, DEVZONE_NONE = 0x07 }
- enum atcacert_transform_e {
  TF_NONE, TF_REVERSE, TF_BIN2HEX_UC, TF_BIN2HEX_LC,
  TF_HEX2BIN_UC, TF_HEX2BIN_LC, TF_BIN2HEX_SPACE_UC, TF_BIN2HEX_SPACE_LC,
  TF_HEX2BIN_SPACE_UC, TF_HEX2BIN_SPACE_LC }

  How to transform the data from the device to the certificate.
- enum atcacert_std_cert_element_e {
  STDCERT_PUBLIC_KEY, STDCERT_SIGNATURE, STDCERT_ISSUE_DATE, STDCERT_EXPIRE_DATE,
  STDCERT_SIGNER_ID, STDCERT_CERT_SN, STDCERT_AUTH_KEY_ID, STDCERT_SUBJ_KEY_ID,
  STDCERT_NUM_ELEMENTS }

Functions

- int atcacert_read_device_loc (const atcacert_device_loc_t *device_loc, uint8_t *data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t *cert_def, const uint8_t ca_public_key[64], uint8_t *cert, size_t *cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size)

  Take a full certificate and write it to the ATECC508A device according to the certificate definition.
- int atcacert_create_csr (const atcacert_def_t *csr_def, uint8_t *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_create_csr_pem (const atcacert_def_t *csr_def, char *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_t response[64])

  Calculates the response to a challenge sent from the host.
- int atcacert_read_subj_key_id (const atcacert_def_t *cert_def, uint8_t subj_key_id[20])

  Reads the subject key ID based on a certificate definition.
- int atcacert_read_cert_size (const atcacert_def_t *cert_def, size_t *cert_size)

  Return the actual certificate size in bytes for a given cert def. Certificate can be variable size, so this gives the absolute buffer size when reading the certificates.
- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t *timestamp, uint8_t *formatted_date, size_t *formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t *formatted_date, size_t formatted_date_size, atcacert_tm_utc_t *timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t *issue_date, uint8_t expire_years, uint8_t enc_dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t *issue_date, atcacert_tm_utc_t *expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t *timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(20)])
- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[(20)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(13)])
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[(13)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(15)])
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[(15)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)
- int atcacert_get_device_locs (const atcacert_def_t *cert_def, atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t *build_state, const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, const uint8_t ca_public_key[64])

Starts the certificate rebuilding process.

- int atcacert_cert_build_process (atcacert_build_state_t ∗build_state, const atcacert_device_loc_t ∗device↩ _loc, const uint8_t ∗device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t ∗build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.
- int atcacert_get_device_data (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const atcacert_device_loc_t ∗device_loc, uint8_t ∗device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.
- int atcacert_set_subj_public_key (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t subj_public_key[64])

  Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_public_key[64])

  Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_key_id[20])

  Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_↩ cert_size, const uint8_t signature[64])

  Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_↩ t signature[64])

  Gets the signature from a certificate.
- int atcacert_set_issue_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_expire_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_↩ t signer_id[2])

  Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_↩ t signer_id[2])

  Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.
- int atcacert_set_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert↩ _size, const uint8_t ∗cert_sn, size_t cert_sn_size)

  Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_↩ t device_sn[9])

  Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.

- int atcacert_get_cert_sn (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t *cert↩ _sn, size_t *cert_sn_size)

    Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t auth_public_key[64])

    Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.
- int atcacert_set_auth_key_id_raw (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t *auth_key_id)

    Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t auth_key_id[20])

    Gets the authority key ID from a certificate.
- int atcacert_set_comp_cert (const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, size_t max_↩ cert_size, const uint8_t comp_cert[72])

    Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.
- int atcacert_get_comp_cert (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_↩ t comp_cert[72])

    Generate the compressed certificate for the given certificate.
- int atcacert_get_tbs (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t **tbs, size_t *tbs_size)

    Get a pointer to the TBS data in a certificate.
- int atcacert_get_tbs_digest (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_↩ t tbs_digest[32])

    Get the SHA256 digest of certificate's TBS data.
- int atcacert_set_cert_element (const atcacert_def_t *cert_def, const atcacert_cert_loc_t *cert_loc, uint8_t *cert, size_t cert_size, const uint8_t *data, size_t data_size)

    Sets an element in a certificate. The data_size must match the size in cert_loc.
- int atcacert_get_cert_element (const atcacert_def_t *cert_def, const atcacert_cert_loc_t *cert_loc, const uint8_t *cert, size_t cert_size, uint8_t *data, size_t data_size)

    Gets an element from a certificate.
- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

    Calculates the key ID for a given public ECC P256 key.
- int atcacert_merge_device_loc (atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_↩ t device_locs_max_count, const atcacert_device_loc_t *device_loc, size_t block_size)

    Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.
- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t *device_loc1, const atcacert_device_loc_t *device_loc2)

    Determines if the two device locations overlap.
- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

    Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.
- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

    Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.
- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t *data, size_t data_size, uint8_t *destination, size_t *destination_size)

    Apply the specified transform to the specified data.
- int atcacert_max_cert_size (const atcacert_def_t *cert_def, size_t *max_cert_size)

    Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.
- int atcacert_der_enc_length (uint32_t length, uint8_t *der_length, size_t *der_length_size)

Encode a length in DER format.
- int atcacert_der_dec_length (const uint8_t *der_length, size_t *der_length_size, uint32_t *length)

  Decode a DER format length.
- int atcacert_der_adjust_length (uint8_t *der_length, size_t *der_length_size, int delta_length, uint32_t *new_length)
- int atcacert_der_enc_integer (const uint8_t *int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t *der_int, size_t *der_int_size)

  Encode an ASN.1 integer in DER format, including tag and length fields.
- int atcacert_der_dec_integer (const uint8_t *der_int, size_t *der_int_size, uint8_t *int_data, size_t *int_data_size)

  Decode an ASN.1 DER encoded integer.
- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t *der_sig, size_t *der_sig_size)

  Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t *der_sig, size_t *der_sig_size, uint8_t raw_sig[64])

  Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.
- int atcacert_verify_cert_hw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using the host's ATECC device for crypto functions.
- int atcacert_verify_cert_sw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [5]

8.5.1 Detailed Description

These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.

8.5.2 Macro Definition Documentation

8.5.2.1 ATCA_PACKED

```
#define ATCA_PACKED
```

8.5.2.2 ATCACERT_DATE_FORMAT_SIZES_COUNT

```
#define ATCACERT_DATE_FORMAT_SIZES_COUNT 5
```

8.5.2.3 ATCACERT_E_BAD_CERT

```
#define ATCACERT_E_BAD_CERT 10
```

Certificate structure is bad in some way.

8.5.2.4 ATCACERT_E_BAD_PARAMS

```
#define ATCACERT_E_BAD_PARAMS 2
```

Invalid/bad parameter passed to function.

8.5.2.5 ATCACERT_E_BUFFER_TOO_SMALL

```
#define ATCACERT_E_BUFFER_TOO_SMALL 3
```

Supplied buffer for output is too small to hold the result.

8.5.2.6 ATCACERT_E_DECODING_ERROR

```
#define ATCACERT_E_DECODING_ERROR 4
```

Data being decoded/parsed has an invalid format.

8.5.2.7 ATCACERT_E_ELEM_MISSING

```
#define ATCACERT_E_ELEM_MISSING 8
```

The certificate element isn't defined for the certificate definition.

8.5.2.8 ATCACERT_E_ELEM_OUT_OF_BOUNDS

```
#define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9
```

Certificate element is out of bounds for the given certificate.

8.5.2.9 ATCACERT_E_ERROR

```
#define ATCACERT_E_ERROR 1
```

General error.

8.5.2.10 ATCACERT_E_INVALID_DATE

```
#define ATCACERT_E_INVALID_DATE 5
```

Date is invalid.

8.5.2.11 ATCACERT_E_INVALID_TRANSFORM

```
#define ATCACERT_E_INVALID_TRANSFORM 13
```

Invalid transform passed to function.

8.5.2.12 ATCACERT_E_SUCCESS

```
#define ATCACERT_E_SUCCESS 0
```

Operation completed successfully.

8.5.2.13 ATCACERT_E_UNEXPECTED_ELEM_SIZE

```
#define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7
```

A certificate element size was not what was expected.

8.5.2.14 ATCACERT_E_UNIMPLEMENTED

```
#define ATCACERT_E_UNIMPLEMENTED 6
```

Function is unimplemented for the current configuration.

8.5.2.15 ATCACERT_E_VERIFY_FAILED

```
#define ATCACERT_E_VERIFY_FAILED 12
```

Certificate or challenge/response verification failed.

8.5.2.16 ATCACERT_E_WRONG_CERT_DEF

```
#define ATCACERT_E_WRONG_CERT_DEF 11
```

8.5.2.17 DATEFMT_ISO8601_SEP

```
#define DATEFMT_ISO8601_SEP 0
```

ISO8601 full date YYYY-MM-DDThh:mm:ssZ.

Date formats.

8.5.2.18 DATEFMT_ISO8601_SEP_SIZE

```
#define DATEFMT_ISO8601_SEP_SIZE (20)
```

8.5.2.19 DATEFMT_MAX_SIZE

```
#define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
```

8.5.2.20 DATEFMT_POSIX_UINT32_BE

```
#define DATEFMT_POSIX_UINT32_BE 2
```

POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, big endian.

8.5.2.21 DATEFMT_POSIX_UINT32_BE_SIZE

```
#define DATEFMT_POSIX_UINT32_BE_SIZE (4)
```

8.5.2.22 DATEFMT_POSIX_UINT32_LE

```
#define DATEFMT_POSIX_UINT32_LE 3
```

POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, little endian.

8.5.2.23 DATEFMT_POSIX_UINT32_LE_SIZE

```
#define DATEFMT_POSIX_UINT32_LE_SIZE (4)
```

8.5.2.24 DATEFMT_RFC5280_GEN

```
#define DATEFMT_RFC5280_GEN 4
```

RFC 5280 (X.509) 4.1.2.5.2 GeneralizedTime format YYYYMMDDhhmmssZ.

8.5.2.25 DATEFMT_RFC5280_GEN_SIZE

```
#define DATEFMT_RFC5280_GEN_SIZE (15)
```

8.5.2.26 DATEFMT_RFC5280_UTC

```
#define DATEFMT_RFC5280_UTC 1
```

RFC 5280 (X.509) 4.1.2.5.1 UTCTime format YYMMDDhhmmssZ.

8.5.2.27 DATEFMT_RFC5280_UTC_SIZE

```
#define DATEFMT_RFC5280_UTC_SIZE (13)
```

8.5.2.28 FALSE

```
#define FALSE (0)
```

8.5.2.29 TRUE

```
#define TRUE (1)
```

8.5.3 Typedef Documentation

8.5.3.1 atcacert_build_state_t

```
typedef struct atcacert_build_state_s atcacert_build_state_t
```

Tracks the state of a certificate as it's being rebuilt from device information.

8.5.3.2 atcacert_cert_element_t

```
typedef struct atcacert_cert_element_s atcacert_cert_element_t
```

Defines a generic dynamic element for a certificate including the device and template locations.

8.5.3.3 atcacert_cert_loc_t

```
typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
```

Defines a chunk of data in a certificate template.

8.5.3.4 atcacert_cert_sn_src_t

```
typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
```

Sources for the certificate serial number.

8.5.3.5 atcacert_cert_type_t

```
typedef enum atcacert_cert_type_e atcacert_cert_type_t
```

Types of certificates.

8.5.3.6 atcacert_date_format_t

```
typedef uint8_t atcacert_date_format_t
```

8.5.3.7 atcacert_def_t

```
typedef struct atcacert_def_s atcacert_def_t
```

Defines a certificate and all the pieces to work with it.

If any of the standard certificate elements (std_cert_elements) are not a part of the certificate definition, set their count to 0 to indicate their absence.

8.5.3.8 atcacert_device_loc_t

```
typedef struct atcacert_device_loc_s atcacert_device_loc_t
```

Defines a chunk of data in an ATECC device.

8.5.3.9 atcacert_device_zone_t

```
typedef enum atcacert_device_zone_e atcacert_device_zone_t
```

ATECC device zones. The values match the Zone Encodings as specified in the datasheet.

8.5.3.10 atcacert_std_cert_element_t

```
typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
```

Standard dynamic certificate elements.

8.5.3.11 atcacert_tm_utc_t

```
typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
```

Holds a broken-down date in UTC. Mimics atcacert_tm_utc_t from time.h.

8.5.3.12 atcacert_transform_t

```
typedef enum atcacert_transform_e atcacert_transform_t
```

How to transform the data from the device to the certificate.

8.5.4 Enumeration Type Documentation

8.5.4.1 atcacert_cert_sn_src_e

```
enum atcacert_cert_sn_src_e
```

Sources for the certificate serial number.

Enumerator

SNSRC_STORED	Cert serial is stored on the device.
SNSRC_STORED_DYNAMIC	Cert serial is stored on the device with the first byte being the DER size (X509 certs only).
SNSRC_DEVICE_SN	Cert serial number is 0x40(MSB) + 9-byte device serial number. Only applies to device certificates.
SNSRC_SIGNER_ID	Cert serial number is 0x40(MSB) + 2-byte signer ID. Only applies to signer certificates.
SNSRC_PUB_KEY_HASH	Cert serial number is the SHA256(Subject public key + Encoded dates), with uppermost 2 bits set to 01.
SNSRC_DEVICE_SN_HASH	Cert serial number is the SHA256(Device SN + Encoded dates), with uppermost 2 bits set to 01. Only applies to device certificates.
SNSRC_PUB_KEY_HASH_POS	Depreciated, don't use. Cert serial number is the SHA256(Subject public key + Encoded dates), with MSBit set to 0 to ensure it's positive.
SNSRC_DEVICE_SN_HASH_POS	Depreciated, don't use. Cert serial number is the SHA256(Device SN + Encoded dates), with MSBit set to 0 to ensure it's positive. Only applies to device certificates.
SNSRC_PUB_KEY_HASH_RAW	Depreciated, don't use. Cert serial number is the SHA256(Subject public key + Encoded dates).
SNSRC_DEVICE_SN_HASH_RAW	Depreciated, don't use. Cert serial number is the SHA256(Device SN + Encoded dates). Only applies to device certificates.

8.5.4.2 atcacert_cert_type_e

`enum atcacert_cert_type_e`

Types of certificates.

Enumerator

CERTTYPE_X509	Standard X509 certificate.
CERTTYPE_CUSTOM	Custom format.

8.5.4.3 atcacert_device_zone_e

`enum atcacert_device_zone_e`

ATECC device zones. The values match the Zone Encodings as specified in the datasheet.

Enumerator

DEVZONE_CONFIG	Configuration zone.
DEVZONE_OTP	One Time Programmable zone.
DEVZONE_DATA	Data zone (slots).
DEVZONE_NONE	Special value used to indicate there is no device location.

8.5.4.4 atcacert_std_cert_element_e

```
enum atcacert_std_cert_element_e
```

Standard dynamic certificate elements.

Enumerator

STDCERT_PUBLIC_KEY	
STDCERT_SIGNATURE	
STDCERT_ISSUE_DATE	
STDCERT_EXPIRE_DATE	
STDCERT_SIGNER_ID	
STDCERT_CERT_SN	
STDCERT_AUTH_KEY_ID	
STDCERT_SUBJ_KEY_ID	
STDCERT_NUM_ELEMENTS	Special item to give the number of elements in this enum.

8.5.4.5 atcacert_transform_e

```
enum atcacert_transform_e
```

How to transform the data from the device to the certificate.

Enumerator

TF_NONE	No transform, data is used byte for byte.
TF_REVERSE	Reverse the bytes (e.g. change endianness)
TF_BIN2HEX_UC	Convert raw binary into ASCII hex, uppercase.
TF_BIN2HEX_LC	Convert raw binary into ASCII hex, lowercase.
TF_HEX2BIN_UC	Convert ASCII hex, uppercase to binary.
TF_HEX2BIN_LC	Convert ASCII hex, lowercase to binary.
TF_BIN2HEX_SPACE_UC	Convert raw binary into ASCII hex, uppercase space between bytes.
TF_BIN2HEX_SPACE_LC	Convert raw binary into ASCII hex, lowercase space between bytes.
TF_HEX2BIN_SPACE_UC	Convert ASCII hex, uppercase with spaces between bytes to binary.
TF_HEX2BIN_SPACE_LC	Convert ASCII hex, lowercase with spaces between bytes to binary.

8.5.5 Function Documentation

8.5.5.1 atcacert_cert_build_finish()

```
int atcacert_cert_build_finish (
            atcacert_build_state_t * build_state )
```

Completes any final certificate processing required after all data from the device has been incorporated.

The final certificate and its size in bytes are contained in the cert and cert_size elements of the build_state structure. This will be the same buffers as supplied to the atcacert_cert_build_start function at the beginning of the certificate rebuilding process.

Parameters

in	*build_state*	Current certificate build state.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.2 atcacert_cert_build_process()

```
int atcacert_cert_build_process (
            atcacert_build_state_t * build_state,
            const atcacert_device_loc_t * device_loc,
            const uint8_t * device_data )
```

Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.

Parameters

in	*build_state*	Current certificate building state.
in	*device_loc*	Device location structure describing where on the device the following data came from.
in	*device_data*	Actual data from the device. It should represent the offset and byte count specified in the device_loc parameter.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.3 atcacert_cert_build_start()

```
int atcacert_cert_build_start (
            atcacert_build_state_t * build_state,
            const atcacert_def_t * cert_def,
```

```
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t ca_public_key[64] )
```

Starts the certificate rebuilding process.

Parameters

out	*build_state*	Structure is initialized to start the certificate building process. Will be passed to the other certificate building functions.
in	*cert_def*	Certificate definition for the certificate being built.
in	*cert*	Buffer to contain the rebuilt certificate.
in	*cert_size*	As input, the size of the cert buffer in bytes. This value will be adjusted to the current/final size of the certificate through the building process.
in	*ca_public_key*	ECC P256 public key of the certificate authority (issuer) for the certificate being built. Set to NULL if the authority key id is not needed, set properly in the cert_def template, or stored on the device as specified in the cert_def cert_elements.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.4 atcacert_create_csr()

```
int atcacert_create_csr (
            const atcacert_def_t * csr_def,
            uint8_t * csr,
            size_t * csr_size )
```

Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

Parameters

in	*csr_def*	CSR definition describing where to find the dynamic CSR information on the device and how to incorporate it into the template.
out	*csr*	Buffer to receive the CSR.
in,out	*csr_size*	As input, the size of the CSR buffer in bytes. As output, the size of the CSR returned in cert in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.5.5.5 atcacert_create_csr_pem()

```
int atcacert_create_csr_pem (
            const atcacert_def_t * csr_def,
            char * csr,
            size_t * csr_size )
```

Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

Parameters

in	*csr_def*	CSR definition describing where to find the dynamic CSR information on the device and how to incorporate it into the template.
out	*csr*	Buffer to received the CSR formatted as PEM.
in,out	*csr_size*	As input, the size of the CSR buffer in bytes. As output, the size of the CSR as PEM returned in cert in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.5.5.6 atcacert_date_dec()

```
int atcacert_date_dec (
            atcacert_date_format_t format,
            const uint8_t * formatted_date,
            size_t formatted_date_size,
            atcacert_tm_utc_t * timestamp )
```

Parse a formatted timestamp according to the specified format.

Parameters

in	*format*	Format to parse the formatted date as.
in	*formatted_date*	Formatted date to be parsed.
in	*formatted_date_size*	Size of the formatted date in bytes.
out	*timestamp*	Parsed timestamp is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.7 atcacert_date_dec_compcert()

```
int atcacert_date_dec_compcert (
            const uint8_t enc_dates[3],
            atcacert_date_format_t expire_date_format,
            atcacert_tm_utc_t * issue_date,
            atcacert_tm_utc_t * expire_date )
```

Decode the issue and expire dates from the format used by the compressed certificate.

Parameters

in	*enc_dates*	Encoded date from the compressed certificate. 3 bytes.
in	*expire_date_format*	Expire date format. Only used to determine max date when no expiration date is specified by the encoded date.
out	*issue_date*	Decoded issue date is returned here.
out	*expire_date*	Decoded expire date is returned here. If there is no expiration date, the expire date will be set to a maximum value for the given expire_date_format.

Returns

0 on success

8.5.5.8 atcacert_date_dec_iso8601_sep()

```
int atcacert_date_dec_iso8601_sep (
            const uint8_t formatted_date[(20)],
            atcacert_tm_utc_t * timestamp )
```

8.5.5.9 atcacert_date_dec_posix_uint32_be()

```
int atcacert_date_dec_posix_uint32_be (
            const uint8_t formatted_date[(4)],
            atcacert_tm_utc_t * timestamp )
```

8.5.5.10 atcacert_date_dec_posix_uint32_le()

```
int atcacert_date_dec_posix_uint32_le (
            const uint8_t formatted_date[(4)],
            atcacert_tm_utc_t * timestamp )
```

8.5.5.11 atcacert_date_dec_rfc5280_gen()

```
int atcacert_date_dec_rfc5280_gen (
            const uint8_t formatted_date[(15)],
            atcacert_tm_utc_t * timestamp )
```

8.5.5.12 atcacert_date_dec_rfc5280_utc()

```
int atcacert_date_dec_rfc5280_utc (
            const uint8_t formatted_date[(13)],
            atcacert_tm_utc_t * timestamp )
```

8.5.5.13 atcacert_date_enc()

```
int atcacert_date_enc (
            atcacert_date_format_t format,
            const atcacert_tm_utc_t * timestamp,
            uint8_t * formatted_date,
            size_t * formatted_date_size )
```

Format a timestamp according to the format type.

Parameters

in	*format*	Format to use.
in	*timestamp*	Timestamp to format.
out	*formatted_date*	Formatted date will be returned in this buffer.
in,out	*formatted_date_size*	As input, the size of the formatted_date buffer. As output, the size of the returned formatted_date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.14 atcacert_date_enc_compcert()

```
int atcacert_date_enc_compcert (
            const atcacert_tm_utc_t * issue_date,
            uint8_t expire_years,
            uint8_t enc_dates[3] )
```

Encode the issue and expire dates in the format used by the compressed certificate.

Parameters

in	*issue_date*	Issue date to encode. Note that minutes and seconds will be ignored.
in	*expire_years*	Expire date is expressed as a number of years past the issue date. 0 should be used if there is no expire date.
out	*enc_dates*	Encoded dates for use in the compressed certificate is returned here. 3 bytes.

Returns

0 on success

8.5.5.15 atcacert_date_enc_iso8601_sep()

```
int atcacert_date_enc_iso8601_sep (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[(20)] )
```

8.5.5.16 atcacert_date_enc_posix_uint32_be()

```
int atcacert_date_enc_posix_uint32_be (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[(4)] )
```

8.5.5.17 atcacert_date_enc_posix_uint32_le()

```
int atcacert_date_enc_posix_uint32_le (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[(4)] )
```

8.5.5.18 atcacert_date_enc_rfc5280_gen()

```
int atcacert_date_enc_rfc5280_gen (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[(15)] )
```

8.5.5.19 atcacert_date_enc_rfc5280_utc()

```
int atcacert_date_enc_rfc5280_utc (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[(13)] )
```

8.5.5.20 atcacert_date_get_max_date()

```
int atcacert_date_get_max_date (
            atcacert_date_format_t format,
            atcacert_tm_utc_t * timestamp )
```

Return the maximum date available for the given format.

Parameters

in	*format*	Format to get the max date for.
out	*timestamp*	Max date is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.21 atcacert_der_adjust_length()

```
int atcacert_der_adjust_length (
            uint8_t * der_length,
            size_t * der_length_size,
            int delta_length,
            uint32_t * new_length )
```

8.5.5.22 atcacert_der_dec_ecdsa_sig_value()

```
int atcacert_der_dec_ecdsa_sig_value (
            const uint8_t * der_sig,
            size_t * der_sig_size,
            uint8_t raw_sig[64] )
```

Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

This will parse the DER encoding of the signatureValue field as found in an X.509 certificate (RFC 5280). x509_sig should include the tag, length, and value. The value of the signatureValue is the DER encoding of the ECDSA-Sig-Value as specified by RFC 5480 and SECG SEC1.

Parameters

`in`	*der_sig*	X.509 format signature (TLV of signatureValue) to be parsed.
`in,out`	*der_sig_size*	As input, size of the der_sig buffer in bytes. As output, size of the DER x.509 signature parsed from the buffer.
`out`	*raw_sig*	Parsed P256 ECDSA signature will be returned in this buffer. Formatted as R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.23 atcacert_der_dec_integer()

```
int atcacert_der_dec_integer (
              const uint8_t * der_int,
              size_t * der_int_size,
              uint8_t * int_data,
              size_t * int_data_size )
```

Decode an ASN.1 DER encoded integer.

X.680 (http://www.itu.int/rec/T-REC-X.680/en) section 19.8, for tag value X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.3, for encoding

Parameters

`in`	*der_int*	DER encoded ASN.1 integer, including the tag and length fields.
`in,out`	*der_int_size*	As input, the size of the der_int buffer in bytes. As output, the size of the DER integer decoded in bytes.
`out`	*int_data*	Decode integer is returned in this buffer in a signed big-endian format.
`in,out`	*int_data_size*	As input, the size of int_data in bytes. As output, the size of the decoded integer in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.24 atcacert_der_dec_length()

```
int atcacert_der_dec_length (
              const uint8_t * der_length,
              size_t * der_length_size,
              uint32_t * length )
```

Decode a DER format length.

X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.1.3, for encoding

Parameters

in	*der_length*	DER encoded length.
in,out	*der_length_size*	As input, the size of the der_length buffer in bytes. As output, the size of the DER encoded length that was decoded.
out	*length*	Decoded length is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.25 atcacert_der_enc_ecdsa_sig_value()

```
int atcacert_der_enc_ecdsa_sig_value (
            const uint8_t raw_sig[64],
            uint8_t * der_sig,
            size_t * der_sig_size )
```

Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.

This will return the DER encoding of the signatureValue field as found in an X.509 certificate (RFC 5280). This include the tag, length, and value. The value of the signatureValue is the DER encoding of the ECDSA-Sig-Value as specified by RFC 5480 and SECG SEC1.

Parameters

in	*raw_sig*	P256 ECDSA signature to be formatted. Input format is R and S integers concatenated together. 64 bytes.
out	*der_sig*	X.509 format signature (TLV of signatureValue) will be returned in this buffer.
in,out	*der_sig_size*	As input, the size of the x509_sig buffer in bytes. As output, the size of the returned X.509 signature in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.26 atcacert_der_enc_integer()

```
int atcacert_der_enc_integer (
            const uint8_t * int_data,
            size_t int_data_size,
            uint8_t is_unsigned,
            uint8_t * der_int,
            size_t * der_int_size )
```

Encode an ASN.1 integer in DER format, including tag and length fields.

X.680 (http://www.itu.int/rec/T-REC-X.680/en) section 19.8, for tag value X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.3, for encoding

Parameters

in	*int_data*	Raw integer in big-endian format.
in	*int_data_size*	Size of the raw integer in bytes.
in	*is_unsigned*	Indicate whether the input integer should be treated as unsigned.
out	*der_int*	DER encoded integer is returned in this buffer.
in,out	*der_int_size*	As input, the size of the der_int buffer in bytes. As output, the size of the DER integer returned in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.27 atcacert_der_enc_length()

```
int atcacert_der_enc_length (
            uint32_t length,
            uint8_t * der_length,
            size_t * der_length_size )
```

Encode a length in DER format.

X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.1.3, for encoding

Parameters

in	*length*	Length to be encoded.
out	*der_length*	DER encoded length will returned in this buffer.
in,out	*der_length_size*	As input, size of der_length buffer in bytes. As output, the size of the DER length encoding in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.28 atcacert_gen_cert_sn()

```
int atcacert_gen_cert_sn (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t device_sn[9] )
```

Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.

This method requires certain elements in the certificate be set properly as they're used for generating the serial number. See atcacert_cert_sn_src_t for what elements should be set in the certificate beforehand. If the sn_source is set to SNSRC_STORED or SNSRC_STORED_DYNAMIC, the function will return ATCACERT_E_SUCCESS without making any changes to the certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*device_sn*	Device serial number, only used if required by the sn_source scheme. Can be set to NULL, if not required.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.29 atcacert_gen_challenge_hw()

```
int atcacert_gen_challenge_hw (
            uint8_t challenge[32] )
```

Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.

Parameters

out	*challenge*	Random challenge is return here. 32 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.30 atcacert_gen_challenge_sw()

```
int atcacert_gen_challenge_sw (
            uint8_t challenge[32] )
```

Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.

Parameters

out	*challenge*	Random challenge is return here. 32 bytes.

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

8.5.5.31 atcacert_get_auth_key_id()

```
int atcacert_get_auth_key_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t auth_key_id[20] )
```

Gets the authority key ID from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*auth_key↵_id*	Authority key ID is returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.32 atcacert_get_cert_element()

```
int atcacert_get_cert_element (
            const atcacert_def_t * cert_def,
            const atcacert_cert_loc_t * cert_loc,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t * data,
            size_t data_size )
```

Gets an element from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert_loc*	Certificate location for this element.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*data*	Element data will be returned in this buffer. This buffer must be large enough to hold cert_loc.count bytes.
in	*data_size*	Expected size of the cert element data.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.33 atcacert_get_cert_sn()

```
int atcacert_get_cert_sn (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t * cert_sn,
            size_t * cert_sn_size )
```

Gets the certificate serial number from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*cert_sn*	Certificate SN will be returned in this buffer.
in,out	*cert_sn_size*	As input, the size of the cert_sn buffer. As output, the size of the certificate SN (cert_sn) in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.34 atcacert_get_comp_cert()

```
int atcacert_get_comp_cert (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t comp_cert[72] )
```

Generate the compressed certificate for the given certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to generate the compressed certificate for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*comp_cert*	Compressed certificate is returned in this buffer. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.35 atcacert_get_device_data()

```
int atcacert_get_device_data (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const atcacert_device_loc_t * device_loc,
            uint8_t * device_data )
```

Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.

The atcacert_add_device_locs function can be used to generate a list of device locations a particular certificate definition requires.

Parameters

in	*cert_def*	Certificate definition for the certificate we're getting data from.
in	*cert*	Certificate to get the device data from.
in	*cert_size*	Size of the certificate in bytes.
in	*device_loc*	Device location to request data for.
out	*device_data*	Buffer that represents the device data in device_loc. Required to be at least device_loc.count in size.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.36 atcacert_get_device_locs()

```
int atcacert_get_device_locs (
            const atcacert_def_t * cert_def,
            atcacert_device_loc_t * device_locs,
            size_t * device_locs_count,
            size_t device_locs_max_count,
            size_t block_size )
```

Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.

The block_size parameter will adjust all added device locations to have a offset and count that aligns with that block size. This allows one to generate a list of device locations that matches specific read or write semantics (e.g. 4 byte or 32 byte reads).

Parameters

in	*cert_def*	Certificate definition containing all the device locations to add to the list.
in,out	*device_locs*	List of device locations to add to.
in,out	*device_locs_count*	As input, existing size of the device locations list. As output, the new size of the device locations list.
in	*device_locs_max_count*	Maximum number of elements device_locs can hold.
in	*block_size*	Block size to align all offsets and counts to when adding device locations.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.37 atcacert_get_expire_date()

```
int atcacert_get_expire_date (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            atcacert_tm_utc_t * timestamp )
```

Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*timestamp*	Expire date is returned in this structure.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.38 atcacert_get_issue_date()

```
int atcacert_get_issue_date (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            atcacert_tm_utc_t * timestamp )
```

Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*timestamp*	Issue date is returned in this structure.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.39 atcacert_get_key_id()

```
int atcacert_get_key_id (
            const uint8_t public_key[64],
            uint8_t key_id[20] )
```

Calculates the key ID for a given public ECC P256 key.

Uses method 1 for calculating the keyIdentifier as specified by RFC 5280, section 4.2.1.2: (1) The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).

Parameters

in	*public_key*	ECC P256 public key to calculate key key ID for. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*key_id*	Calculated key ID will be returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.40 atcacert_get_response()

```
int atcacert_get_response (
            uint8_t device_private_key_slot,
            const uint8_t challenge[32],
            uint8_t response[64] )
```

Calculates the response to a challenge sent from the host.

The challenge-response protocol is an ECDSA Sign and Verify. This performs the ECDSA Sign on the challenge and returns the signature as the response.

Parameters

in	*device_private_key_slot*	Slot number for the device's private key. This must be the same slot used to generate the public key included in the device's certificate.
in	*challenge*	Challenge to generate the response for. Must be 32 bytes.
out	*response*	Response will be returned in this buffer. 64 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.5.5.41 atcacert_get_signature()

```
int atcacert_get_signature (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t signature[64] )
```

Gets the signature from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*signature*	Signature is returned in this buffer. Formatted at R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.42 atcacert_get_signer_id()

```
int atcacert_get_signer_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t signer_id[2] )
```

Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*signer←_id*	Signer ID will be returned in this buffer. 2 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.43 atcacert_get_subj_key_id()

```
int atcacert_get_subj_key_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t subj_key_id[20] )
```

Gets the subject key ID from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*subj_key←_id*	Subject key ID is returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.44 atcacert_get_subj_public_key()

```
int atcacert_get_subj_public_key (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t subj_public_key[64] )
```

Gets the subject public key from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*subj_public_key*	Subject public key is returned in this buffer. Formatted at X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.45 atcacert_get_tbs()

```
int atcacert_get_tbs (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ** tbs,
            size_t * tbs_size )
```

Get a pointer to the TBS data in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get the TBS data pointer for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*tbs*	Pointer to a const pointer that will be set the start of the TBS data.
out	*tbs_size*	Size of the TBS data will be returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.46 atcacert_get_tbs_digest()

```
int atcacert_get_tbs_digest (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t tbs_digest[32] )
```

Get the SHA256 digest of certificate's TBS data.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get the TBS data pointer for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*tbs_digest*	TBS data digest will be returned here. 32 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.47 atcacert_is_device_loc_overlap()

```
int atcacert_is_device_loc_overlap (
            const atcacert_device_loc_t * device_loc1,
            const atcacert_device_loc_t * device_loc2 )
```

Determines if the two device locations overlap.

Parameters

in	*device_loc1*	First device location to check.
in	*device_loc2*	Second device location o check.

Returns

0 (false) if they don't overlap, non-zero if the do overlap.

8.5.5.48 atcacert_max_cert_size()

```
int atcacert_max_cert_size (
            const atcacert_def_t * cert_def,
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

Parameters

in	*cert_def*	Certificate definition to find a max size for.
out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.49 atcacert_merge_device_loc()

```
int atcacert_merge_device_loc (
            atcacert_device_loc_t * device_locs,
            size_t * device_locs_count,
            size_t device_locs_max_count,
            const atcacert_device_loc_t * device_loc,
            size_t block_size )
```

Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.

The block_size parameter will adjust all added device locations to have an offset and count that aligns with that block size. This allows one to generate a list of device locations that matches specific read/write semantics (e.g. 4 byte or 32 byte reads). Note that this block_size only applies to the device_loc being added. Existing device locations in the list won't be modified to match the block size.

Parameters

in,out	*device_locs*	Existing device location list to merge the new device location into.
in,out	*device_locs_count*	As input, the existing number of items in the device_locs list. As output, the new size of the device_locs list.
in	*device_locs_max_count*	Maximum number of items the device_locs list can hold.
in	*device_loc*	New device location to be merged into the device_locs list.
in	*block_size*	Block size to align all offsets and counts to when adding device location.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.50 atcacert_public_key_add_padding()

```
void atcacert_public_key_add_padding (
            const uint8_t raw_key[64],
            uint8_t padded_key[72] )
```

Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.

Parameters

in	*raw_key*	Public key as X and Y integers concatenated together. 64 bytes.
out	*padded_key*	Padded key is returned in this buffer. X and Y integers are padded with 4 bytes of 0 in the MSB. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.51 atcacert_public_key_remove_padding()

```
void atcacert_public_key_remove_padding (
            const uint8_t padded_key[72],
            uint8_t raw_key[64] )
```

Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.

Parameters

out	*padded_key*	X and Y integers are padded with 4 bytes of 0 in the MSB. 72 bytes.
in	*raw_key*	Raw key is returned in this buffer. Public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.52 atcacert_read_cert()

```
int atcacert_read_cert (
            const atcacert_def_t * cert_def,
            const uint8_t ca_public_key[64],
            uint8_t * cert,
            size_t * cert_size )
```

Reads the certificate specified by the certificate definition from the ATECC508A device.

This process involves reading the dynamic cert data from the device and combining it with the template found in the certificate definition.

Parameters

in	*cert_def*	Certificate definition describing where to find the dynamic certificate information on the device and how to incorporate it into the template.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total). Set to NULL if the authority key id is not needed, set properly in the cert_def template, or stored on the device as specifed in the cert_def cert_elements.
out	*cert*	Buffer to received the certificate.
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.53 atcacert_read_cert_size()

```
int atcacert_read_cert_size (
            const atcacert_def_t * cert_def,
            size_t * cert_size )
```

Return the actual certificate size in bytes for a given cert def. Certificate can be variable size, so this gives the absolute buffer size when reading the certificates.

Parameters

in	*cert_def*	Certificate definition to find a max size for.
out	*cert_size*	Certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.54 atcacert_read_device_loc()

```
int atcacert_read_device_loc (
            const atcacert_device_loc_t * device_loc,
            uint8_t * data )
```

Read the data from a device location.

Parameters

in	*device_loc*	Device location to read data from.
out	*data*	Data read is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.55 atcacert_read_subj_key_id()

```
int atcacert_read_subj_key_id (
            const atcacert_def_t * cert_def,
            uint8_t subj_key_id[20] )
```

Reads the subject key ID based on a certificate definition.

Parameters

in	*cert_def*	Certificate definition
out	*subj_key_id*	Subject key ID is returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.56 atcacert_set_auth_key_id()

```
int atcacert_set_auth_key_id (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t auth_public_key[64] )
```

Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*auth_public_key*	Authority public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.57 atcacert_set_auth_key_id_raw()

```
int atcacert_set_auth_key_id_raw (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t * auth_key_id )
```

Sets the authority key ID in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*auth_key_id*	Authority key ID. Same size as defined in the cert_def.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.58 atcacert_set_cert_element()

```
int atcacert_set_cert_element (
            const atcacert_def_t * cert_def,
            const atcacert_cert_loc_t * cert_loc,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t * data,
            size_t data_size )
```

Sets an element in a certificate. The data_size must match the size in cert_loc.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert_loc*	Certificate location for this element.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*data*	Element data to insert into the certificate. Buffer must contain cert_loc.count bytes to be copied into the certificate.
in	*data_size*	Size of the data in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.59 atcacert_set_cert_sn()

```
int atcacert_set_cert_sn (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            size_t max_cert_size,
            const uint8_t * cert_sn,
            size_t cert_sn_size )
```

Sets the certificate serial number in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	Size of the certificate (cert) in bytes.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*cert_sn*	Certificate serial number.
in	*cert_sn_size*	Size of the certificate serial number in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.60 atcacert_set_comp_cert()

```
int atcacert_set_comp_cert (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            size_t max_cert_size,
            const uint8_t comp_cert[72] )
```

Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	As input, size of the certificate (cert) in bytes. As output, the new size of the certificate.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*comp_cert*	Compressed certificate. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success. ATCACERT_E_WRONG_CERT_DEF if the template ID, chain ID, and/or SN source don't match between the cert_def and the compressed certificate.

8.5.5.61 atcacert_set_expire_date()

```
int atcacert_set_expire_date (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const atcacert_tm_utc_t * timestamp )
```

Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*timestamp*	Expire date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.62 atcacert_set_issue_date()

```
int atcacert_set_issue_date (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const atcacert_tm_utc_t * timestamp )
```

Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*timestamp*	Issue date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.63 atcacert_set_signature()

```
int atcacert_set_signature (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            size_t max_cert_size,
            const uint8_t signature[64] )
```

Sets the signature in a certificate. This may alter the size of the X.509 certificates.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	As input, size of the certificate (cert) in bytes. As output, the new size of the certificate.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*signature*	Signature as R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.64 atcacert_set_signer_id()

```
int atcacert_set_signer_id (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t signer_id[2] )
```

Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*signer_id*	Signer ID.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.65 atcacert_set_subj_public_key()

```
int atcacert_set_subj_public_key (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t subj_public_key[64] )
```

Sets the subject public key and subject key ID in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*subj_public_key*	Subject public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.66 atcacert_transform_data()

```
int atcacert_transform_data (
            atcacert_transform_t transform,
            const uint8_t * data,
            size_t data_size,
            uint8_t * destination,
            size_t * destination_size )
```

Apply the specified transform to the specified data.

Parameters

in	*transform*	Transform to be performed.
in	*data*	Input data to be transformed.
in	*data_size*	Size of the input data in bytes.
out	*destination*	Destination buffer to hold the transformed data.
in,out	*destination_size*	As input, the size of the destination buffer. As output the size of the transformed data.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.5.67 atcacert_verify_cert_hw()

```
int atcacert_verify_cert_hw (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ca_public_key[64] )
```

Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.

Parameters

in	*cert_def*	Certificate definition describing how to extract the TBS and signature components from the certificate specified.
in	*cert*	Certificate to verify.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total).

Returns

ATCACERT_E_SUCCESS if the verify succeeds, ATCACERT_VERIFY_FAILED or ATCA_EXECUTION_↩ ERROR if it fails to verify. ATCA_EXECUTION_ERROR may occur when the public key is invalid and doesn't fall on the P256 curve.

8.5.5.68 atcacert_verify_cert_sw()

```
int atcacert_verify_cert_sw (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ca_public_key[64] )
```

Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.

Parameters

in	*cert_def*	Certificate definition describing how to extract the TBS and signature components from the certificate specified.
in	*cert*	Certificate to verify.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total).

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

8.5.5.69 atcacert_verify_response_hw()

```
int atcacert_verify_response_hw (
            const uint8_t device_public_key[64],
            const uint8_t challenge[32],
            const uint8_t response[64] )
```

Verify a client's response to a challenge using the host's ATECC device for crypto functions.

The challenge-response protocol is an ECDSA Sign and Verify. This performs an ECDSA verify on the response returned by the client, verifying the client has the private key counter-part to the public key returned in its certificate.

Parameters

in	*device_public_key*	Device public key as read from its certificate. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*challenge*	Challenge that was sent to the client. 32 bytes.
in	*response*	Response returned from the client to be verified. 64 bytes.

Returns

ATCACERT_E_SUCCESS if the verify succeeds, ATCACERT_VERIFY_FAILED or ATCA_EXECUTION_↩ERROR if it fails to verify. ATCA_EXECUTION_ERROR may occur when the public key is invalid and doesn't fall on the P256 curve.

8.5.5.70 atcacert_verify_response_sw()

```
int atcacert_verify_response_sw (
            const uint8_t device_public_key[64],
            const uint8_t challenge[32],
            const uint8_t response[64] )
```

Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

The challenge-response protocol is an ECDSA Sign and Verify. This performs an ECDSA verify on the response returned by the client, verifying the client has the private key counter-part to the public key returned in its certificate.

Parameters

in	*device_public_key*	Device public key as read from its certificate. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*challenge*	Challenge that was sent to the client. 32 bytes.
in	*response*	Response returned from the client to be verified. 64 bytes.

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

8.5.5.71 atcacert_write_cert()

```
int atcacert_write_cert (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size )
```

Take a full certificate and write it to the ATECC508A device according to the certificate definition.

Parameters

in	*cert_def*	Certificate definition describing where the dynamic certificate information is and how to store it on the device.
in	*cert*	Full certificate to be stored.
in	*cert_size*	Size of the full certificate in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.5.6 Variable Documentation

8.5.6.1 ATCACERT_DATE_FORMAT_SIZES

```
const size_t ATCACERT_DATE_FORMAT_SIZES[5]  [extern]
```

8.6 Basic Crypto API methods for CryptoAuth Devices (calib_)

These methods provide a simple API to CryptoAuth chips.

8.6.0.1 calib directory - Purpose

The purpose of this directory is to contain the files implementing the APIs for a basic interface to the core Crypto↩ AuthLib library.

High-level functions like these make it very convenient to use the library when standard configurations and defaults are in play. They are the easiest to use when developing examples or trying to understand the "flow" of an authentication operation without getting overwhelmed by the details.

This makes simple jobs easy and if you need more sophistication and power, you can employ the full power of the CryptoAuthLib object model.

See the Doxygen documentation in cryptoauthlib/docs for details on the API of the calib commands.

Data Structures

- struct atca_sha256_ctx

Typedefs

- typedef struct atca_sha256_ctx atca_sha256_ctx_t
- typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t

Functions

- ATCA_STATUS calib_wakeup (ATCADevice device)

  wakeup the CryptoAuth device
- ATCA_STATUS calib_idle (ATCADevice device)

  idle the CryptoAuth device
- ATCA_STATUS calib_sleep (ATCADevice device)

  invoke sleep on the CryptoAuth device
- ATCA_STATUS _calib_exit (ATCADevice device)

  common cleanup code which idles the device after any operation
- ATCA_STATUS calib_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS calib_get_zone_size (ATCADevice device, uint8_t zone, uint16_t slot, size_t ∗size)

  Gets the size of the specified zone in bytes.
- ATCA_STATUS calib_ecc204_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset for ECC204 device.
- ATCA_STATUS calib_is_locked (ATCADevice device, uint8_t zone, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS calib_is_slot_locked (ATCADevice device, uint16_t slot, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS calib_is_private (ATCADevice device, uint16_t slot, bool ∗is_private)

Check if a slot is a private key.

- ATCA_STATUS calib_ecc204_is_locked (ATCADevice device, uint8_t zone, bool *is_locked)
- ATCA_STATUS calib_ecc204_is_data_locked (ATCADevice device, bool *is_locked)
- ATCA_STATUS calib_ecc204_is_config_locked (ATCADevice device, bool *is_locked)
- ATCA_STATUS calib_aes (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t *aes_in, uint8←_t *aes_out)

  Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS calib_aes_encrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_t *ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS calib_aes_decrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS calib_aes_gfm (ATCADevice device, const uint8_t *h, const uint8_t *input, uint8_t *output)

  Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS calib_checkmac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_←t *challenge, const uint8_t *response, const uint8_t *other_data)

  Compares a MAC response with input values.
- ATCA_STATUS calib_counter (ATCADevice device, uint8_t mode, uint16_t counter_id, uint32_t *counter←_value)

  Compute the Counter functions.
- ATCA_STATUS calib_counter_increment (ATCADevice device, uint16_t counter_id, uint32_t *counter_←value)

  Increments one of the device's monotonic counters.
- ATCA_STATUS calib_counter_read (ATCADevice device, uint16_t counter_id, uint32_t *counter_value)

  Read one of the device's monotonic counters.
- ATCA_STATUS calib_derivekey (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t *mac)

  Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.
- ATCA_STATUS calib_ecdh_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_←t *public_key, uint8_t *pms, uint8_t *out_nonce)

  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS calib_ecdh (ATCADevice device, uint16_t key_id, const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_enc (ATCADevice device, uint16_t key_id, const uint8_t *public_key, uint8_←t *pms, const uint8_t *read_key, uint16_t read_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_ecdh_ioenc (ATCADevice device, uint16_t key_id, const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS calib_ecdh_tempkey (ATCADevice device, const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_tempkey_ioenc (ATCADevice device, const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS calib_gendig (ATCADevice device, uint8_t zone, uint16_t key_id, const uint8_t *other_data, uint8_t other_data_size)

  Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS calib_genkey_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_←t *other_data, uint8_t *public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.

- ATCA_STATUS calib_genkey (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS calib_get_pubkey (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS calib_genkey_mac (ATCADevice device, uint8_t ∗public_key, uint8_t ∗mac)

  Uses Genkey command to calculate SHA256 digest MAC of combining public key and session key.
- ATCA_STATUS calib_hmac (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t ∗digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS calib_info_base (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS calib_info (ATCADevice device, uint8_t ∗revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS calib_info_set_latch (ATCADevice device, bool state)

  Use the Info command to set the persistent latch state for an ATECC608 device.
- ATCA_STATUS calib_info_get_latch (ATCADevice device, bool ∗state)

  Use the Info command to get the persistent latch current state for an ATECC608 device.
- ATCA_STATUS calib_info_privkey_valid (ATCADevice device, uint16_t key_id, uint8_t ∗is_valid)

  Use Info command to check ECC Private key stored in key slot is valid or not.
- ATCA_STATUS calib_info_lock_status (ATCADevice device, uint16_t param2, uint8_t ∗is_locked)
- ATCA_STATUS calib_kdf (ATCADevice device, uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

  Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS calib_lock (ATCADevice device, uint8_t mode, uint16_t summary_crc)

  The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS calib_lock_config_zone (ATCADevice device)

  Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS calib_lock_config_zone_crc (ATCADevice device, uint16_t summary_crc)

  Lock the config zone with summary CRC.
- ATCA_STATUS calib_lock_data_zone (ATCADevice device)

  Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS calib_lock_data_zone_crc (ATCADevice device, uint16_t summary_crc)

  Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS calib_lock_data_slot (ATCADevice device, uint16_t slot)

  Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).
- ATCA_STATUS calib_ecc204_lock_config_slot (ATCADevice device, uint8_t slot, uint16_t summary_crc)

  Use Lock command to lock individual configuration zone slots.
- ATCA_STATUS calib_ecc204_lock_config_zone (ATCADevice device)

  Use lock command to lock complete configuration zone.
- ATCA_STATUS calib_ecc204_lock_data_slot (ATCADevice device, uint8_t slot)

  Use lock command to lock data zone slot.
- ATCA_STATUS calib_mac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS calib_nonce_base (ATCADevice device, uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.

- ATCA_STATUS calib_nonce (ATCADevice device, const uint8_t *num_in)

    Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_nonce_load (ATCADevice device, uint8_t target, const uint8_t *num_in, uint16_t num_in_size)

    Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS calib_nonce_rand (ATCADevice device, const uint8_t *num_in, uint8_t *rand_out)

    Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_challenge (ATCADevice device, const uint8_t *num_in)

    Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_challenge_seed_update (ATCADevice device, const uint8_t *num_in, uint8_t *rand_out)

    Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_nonce_gen_session_key (ATCADevice device, uint16_t param2, uint8_t *num_in, uint8_t *rand_out)

    Use Nonce command to generate session key for use by a subsequent write command This Mode only supports in ECC204 device.
- ATCA_STATUS calib_priv_write (ATCADevice device, uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[(20)])
- ATCA_STATUS calib_random (ATCADevice device, uint8_t *rand_out)

    Executes Random command, which generates a 32 byte random number from the CryptoAuth device.
- ATCA_STATUS calib_read_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t *data, uint8_t len)

    Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS calib_read_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset, uint8_t *data, size_t length)

    Used to read an arbitrary number of bytes from any zone configured for clear reads.
- ATCA_STATUS calib_read_serial_number (ATCADevice device, uint8_t *serial_number)

    Executes Read command, which reads the 9 byte serial number of the device from the config zone.
- ATCA_STATUS calib_read_pubkey (ATCADevice device, uint16_t slot, uint8_t *public_key)

    Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS calib_read_sig (ATCADevice device, uint16_t slot, uint8_t *sig)

    Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS calib_read_config_zone (ATCADevice device, uint8_t *config_data)

    Executes Read command to read the complete device configuration zone.
- ATCA_STATUS calib_cmp_config_zone (ATCADevice device, uint8_t *config_data, bool *same_config)

    Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS calib_ecc204_read_zone (ATCADevice device, uint8_t zone, uint8_t slot, uint8_t block, size_t offset, uint8_t *data, uint8_t len)
- ATCA_STATUS calib_ecc204_read_config_zone (ATCADevice device, uint8_t *config_data)
- ATCA_STATUS calib_ecc204_read_serial_number (ATCADevice device, uint8_t *serial_number)
- ATCA_STATUS calib_ecc204_read_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t block, uint8_t *data, size_t length)
- ATCA_STATUS calib_ecc204_cmp_config_zone (ATCADevice device, uint8_t *config_data, bool *same_config)
- ATCA_STATUS calib_read_enc (ATCADevice device, uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_secureboot (ATCADevice device, uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

    Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS calib_secureboot_mac (ATCADevice device, uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS calib_selftest (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t *result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the AT←ECC608 chip.
- ATCA_STATUS calib_sha_base (ATCADevice device, uint8_t mode, uint16_t length, const uint8_t *data_in, uint8_t *data_out, uint16_t *data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS calib_sha_start (ATCADevice device)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS calib_sha_update (ATCADevice device, const uint8_t *message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS calib_sha_end (ATCADevice device, uint8_t *digest, uint16_t length, const uint8_←t *message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_read_context (ATCADevice device, uint8_t *context, uint16_t *context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS calib_sha_write_context (ATCADevice device, const uint8_t *context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.
- ATCA_STATUS calib_sha (ATCADevice device, uint16_t length, const uint8_t *message, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_hw_sha2_256 (ATCADevice device, const uint8_t *data, size_t data_size, uint8_←t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_hw_sha2_256_init (ATCADevice device, atca_sha256_ctx_t *ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS calib_hw_sha2_256_update (ATCADevice device, atca_sha256_ctx_t *ctx, const uint8_←t *data, size_t data_size)

  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS calib_hw_sha2_256_finish (ATCADevice device, atca_sha256_ctx_t *ctx, uint8_t *digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS calib_sha_hmac_init (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, uint16_t key_←slot)

  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_update (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, const uint8←_t *data, size_t data_size)

  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_finish (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, uint8_←t *digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac (ATCADevice device, const uint8_t *data, size_t data_size, uint16_t key←_slot, uint8_t *digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS calib_sign_base (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t *signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS calib_sign (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_sign_internal (ATCADevice device, uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t *signature)

Executes Sign command to sign an internally generated message.

- ATCA_STATUS calib_ecc204_sign (ATCADevice device, uint16_t key_id, const uint8_t ∗msg, uint8_↩ t ∗signature)

  Execute sign command to sign the 32 bytes message digest using private key mentioned in slot.
- ATCA_STATUS calib_updateextra (ATCADevice device, uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).
- ATCA_STATUS calib_verify (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS calib_verify_extern (ATCADevice device, const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_verify_extern_mac (ATCADevice device, const uint8_t ∗message, const uint8_↩ t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_stored (ATCADevice device, const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩ Key for other devices.
- ATCA_STATUS calib_verify_stored_mac (ATCADevice device, const uint8_t ∗message, const uint8_↩ t ∗signature, uint16_t key_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_validate (ATCADevice device, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS calib_verify_invalidate (ATCADevice device, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS calib_write (ATCADevice device, uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS calib_write_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_↩ t offset, const uint8_t ∗data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS calib_write_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS calib_write_pubkey (ATCADevice device, uint16_t slot, const uint8_t ∗public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS calib_write_config_zone (ATCADevice device, const uint8_t ∗config_data)

  Executes the Write command, which writes the configuration zone.
- ATCA_STATUS calib_ecc204_write (ATCADevice device, uint8_t zone, uint16_t address, const uint8_↩ t ∗value, const uint8_t ∗mac)

- ATCA_STATUS calib_ecc204_write_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)
- ATCA_STATUS calib_ecc204_write_config_zone (ATCADevice device, const uint8_t ∗config_data)
- ATCA_STATUS calib_ecc204_write_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_↩ t block, const uint8_t ∗data, size_t length)
- ATCA_STATUS calib_write_enc (ATCADevice device, uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_ecc204_write_enc (ATCADevice device, uint8_t slot, uint8_t ∗data, uint8_↩ t ∗transport_key, uint8_t key_id, uint8_t num_in[(20)])
- ATCA_STATUS calib_write_config_counter (ATCADevice device, uint16_t counter_id, uint32_t counter_↩ value)

  Initialize one of the monotonic counters in device with a specific value.

- const char ∗ atca_basic_aes_gcm_version = "2.0"

8.6.1 Detailed Description

These methods provide a simple API to CryptoAuth chips.

8.6.2 Typedef Documentation

8.6.2.1 atca_hmac_sha256_ctx_t

```
typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t
```

8.6.2.2 atca_sha256_ctx_t

```
typedef struct atca_sha256_ctx atca_sha256_ctx_t
```

8.6.3 Function Documentation

8.6.3.1 _calib_exit()

```
ATCA_STATUS _calib_exit (
            ATCADevice device )
```

common cleanup code which idles the device after any operation

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.2 calib_aes()

```
ATCA_STATUS calib_aes (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * aes_in,
            uint8_t * aes_out )
```

Compute the AES-128 encrypt, decrypt, or GFM calculation.

Parameters

in	*device*	Device context pointer
in	*mode*	The mode for the AES command.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*aes_in*	Input data to the AES command (16 bytes).
out	*aes_out*	Output data from the AES command is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.3 calib_aes_decrypt()

```
ATCA_STATUS calib_aes_decrypt (
            ATCADevice device,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Perform an AES-128 decrypt operation with a key in the device.

Parameters

in	*device*	Device context pointer
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*ciphertext*	Input ciphertext to be decrypted (16 bytes).
out	*plaintext*	Output plaintext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.4 calib_aes_encrypt()

```
ATCA_STATUS calib_aes_encrypt (
            ATCADevice device,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Perform an AES-128 encrypt operation with a key in the device.

Parameters

in	*device*	Device context pointer
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*plaintext*	Input plaintext to be encrypted (16 bytes).
out	*ciphertext*	Output ciphertext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.5 calib_aes_gfm()

```
ATCA_STATUS calib_aes_gfm (
            ATCADevice device,
            const uint8_t * h,
            const uint8_t * input,
            uint8_t * output )
```

Perform a Galois Field Multiply (GFM) operation.

Parameters

in	*device*	Device context pointer
in	*h*	First input value (16 bytes).
in	*input*	Second input value (16 bytes).
out	*output*	GFM result is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.6 calib_challenge()

```
ATCA_STATUS calib_challenge (
            ATCADevice device,
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*device*	Device context pointer
in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.7 calib_challenge_seed_update()

```
ATCA_STATUS calib_challenge_seed_update (
            ATCADevice device,
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.

Parameters

in	*device*	Device context pointer
in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.8 calib_checkmac()

```
ATCA_STATUS calib_checkmac (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * challenge,
            const uint8_t * response,
            const uint8_t * other_data )
```

Compares a MAC response with input values.

Parameters

in	*device*	Device context pointer
in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key location in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge data (32 bytes)
in	*response*	MAC response data (32 bytes)
in	*other_data*	OtherData parameter (13 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.9 calib_cmp_config_zone()

```
ATCA_STATUS calib_cmp_config_zone (
            ATCADevice device,
            uint8_t * config_data,
            bool * same_config )
```

Compares a specified configuration zone with the configuration zone currently on the device.

This only compares the static portions of the configuration zone and skips those that are unique per device (first 16 bytes) and areas that can change after the configuration zone has been locked (e.g. LastKeyUse).

Parameters

in	*device*	Device context pointer
in	*config_data*	Full configuration data to compare the device against.
out	*same_config*	Result is returned here. True if the static portions on the configuration zones are the same.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Max for all configs

8.6.3.10 calib_counter()

```
ATCA_STATUS calib_counter (
            ATCADevice device,
            uint8_t mode,
            uint16_t counter_id,
            uint32_t * counter_value )
```

Compute the Counter functions.

Parameters

in	*device*	Device context pointer
in	*mode*	the mode used for the counter
in	*counter_id*	The counter to be used
out	*counter_value*	pointer to the counter value returned from device

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.11 calib_counter_increment()

```
ATCA_STATUS calib_counter_increment (
            ATCADevice device,
            uint16_t counter_id,
            uint32_t * counter_value )
```

Increments one of the device's monotonic counters.

Parameters

in	*device*	Device context pointer
in	*counter_id*	Counter to be incremented
out	*counter_value*	New value of the counter is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.12 calib_counter_read()

```
ATCA_STATUS calib_counter_read (
            ATCADevice device,
            uint16_t counter_id,
            uint32_t * counter_value )
```

Read one of the device's monotonic counters.

Parameters

in	*device*	Device context pointer
in	*counter_id*	Counter to be read
out	*counter_value*	Counter value is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.13 calib_derivekey()

```
ATCA_STATUS calib_derivekey (
            ATCADevice device,
            uint8_t mode,
            uint16_t target_key,
            const uint8_t * mac )
```

Executes the DeviceKey command for deriving a new key from a nonce (TempKey) and an existing key.

Parameters

in	*device*	Device context pointer
in	*mode*	Bit 2 must match the value in TempKey.SourceFlag
in	*target_key*	Key slot to be written
in	*mac*	Optional 32 byte MAC used to validate operation. NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.14 calib_ecc204_cmp_config_zone()

```
ATCA_STATUS calib_ecc204_cmp_config_zone (
            ATCADevice device,
            uint8_t * config_data,
            bool * same_config )
```

8.6.3.15 calib_ecc204_get_addr()

```
ATCA_STATUS calib_ecc204_get_addr (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            uint16_t * addr )
```

Compute the address given the zone, slot, block, and offset for ECC204 device.

Parameters

in	*zone*	Zone to get address from. Config(1) or Data(0) which requires a slot.
in	*slot*	Slot Id number for data zone and zero for other zones.
in	*block*	Block number within the data zone .
in	*offset*	Aalways zero.
out	*addr*	Pointer to the address of data or configuration zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.16 calib_ecc204_is_config_locked()

```
ATCA_STATUS calib_ecc204_is_config_locked (
            ATCADevice device,
            bool * is_locked )
```

8.6.3.17 calib_ecc204_is_data_locked()

```
ATCA_STATUS calib_ecc204_is_data_locked (
            ATCADevice device,
            bool * is_locked )
```

8.6.3.18 calib_ecc204_is_locked()

```
ATCA_STATUS calib_ecc204_is_locked (
            ATCADevice device,
            uint8_t zone,
            bool * is_locked )
```

8.6.3.19 calib_ecc204_lock_config_slot()

```
ATCA_STATUS calib_ecc204_lock_config_slot (
            ATCADevice device,
            uint8_t slot,
            uint16_t summary_crc )
```

Use Lock command to lock individual configuration zone slots.

Parameters

in	*device*	Device context pointer
in	*slot*	The slot number to be locked
in	*summary_crc*	CRC calculated over all 16 bytes within the selected slot of the configuration zone.

Returns

ATCA_SUCCESS on success, otherwise an error code

8.6.3.20 calib_ecc204_lock_config_zone()

```
ATCA_STATUS calib_ecc204_lock_config_zone (
            ATCADevice device )
```

Use lock command to lock complete configuration zone.

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code

8.6.3.21 calib_ecc204_lock_data_slot()

```
ATCA_STATUS calib_ecc204_lock_data_slot (
            ATCADevice device,
            uint8_t slot )
```

Use lock command to lock data zone slot.

Parameters

in	*device*	Device context pointer
in	*slot*	The slot number to be locked

Returns

ATCA_SUCCESS on success, otherwise an error code

8.6.3.22 calib_ecc204_read_bytes_zone()

```
ATCA_STATUS calib_ecc204_read_bytes_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            size_t block,
            uint8_t * data,
            size_t length )
```

8.6.3.23 calib_ecc204_read_config_zone()

```
ATCA_STATUS calib_ecc204_read_config_zone (
            ATCADevice device,
            uint8_t * config_data )
```

8.6.3.24 calib_ecc204_read_serial_number()

```
ATCA_STATUS calib_ecc204_read_serial_number (
            ATCADevice device,
            uint8_t * serial_number )
```

8.6.3.25 calib_ecc204_read_zone()

```
ATCA_STATUS calib_ecc204_read_zone (
            ATCADevice device,
            uint8_t zone,
            uint8_t slot,
            uint8_t block,
            size_t offset,
            uint8_t * data,
            uint8_t len )
```

8.6.3.26 calib_ecc204_sign()

```
ATCA_STATUS calib_ecc204_sign (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * msg,
            uint8_t * signature )
```

Execute sign command to sign the 32 bytes message digest using private key mentioned in slot.

Parameters

in	*device*	Device context pointer
in	*key_id*	points to private key slot
in	*msg*	32 bytes message digest
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code

8.6.3.27 calib_ecc204_write()

```
ATCA_STATUS calib_ecc204_write (
            ATCADevice device,
            uint8_t zone,
            uint16_t address,
            const uint8_t * value,
            const uint8_t * mac )
```

8.6.3.28 calib_ecc204_write_bytes_zone()

```
ATCA_STATUS calib_ecc204_write_bytes_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            size_t block,
            const uint8_t * data,
            size_t length )
```

8.6.3.29 calib_ecc204_write_config_zone()

```
ATCA_STATUS calib_ecc204_write_config_zone (
            ATCADevice device,
            const uint8_t * config_data )
```

8.6.3.30 calib_ecc204_write_enc()

```
ATCA_STATUS calib_ecc204_write_enc (
            ATCADevice device,
            uint8_t slot,
            uint8_t * data,
            uint8_t * transport_key,
            uint8_t key_id,
            uint8_t num_in[(20)] )
```

8.6.3.31 calib_ecc204_write_zone()

```
ATCA_STATUS calib_ecc204_write_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            const uint8_t * data,
            uint8_t len )
```

8.6.3.32 calib_ecdh()

```
ATCA_STATUS calib_ecdh (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in a slot and the premaster secret is returned in the clear.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here. 32 bytes.

Returns

ATCA_SUCCESS on success

8.6.3.33 calib_ecdh_base()

```
ATCA_STATUS calib_ecdh_base (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            uint8_t * out_nonce )
```

Base function for generating premaster secret key using ECDH.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode to be used for ECDH computation
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH pre-master secret is returned here (32 bytes) if returned directly. Otherwise NULL.
out	*out_nonce*	Nonce used to encrypt pre-master secret. NULL if output encryption not used.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.34 calib_ecdh_enc()

```
ATCA_STATUS calib_ecdh_enc (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * read_key,
            uint16_t read_key_id,
            const uint8_t num_in[(20)] )
```

8.6.3.35 calib_ecdh_ioenc()

```
ATCA_STATUS calib_ecdh_ioenc (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.36 calib_ecdh_tempkey()

```
ATCA_STATUS calib_ecdh_tempkey (
            ATCADevice device,
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in TempKey and the premaster secret is returned in the clear.

Parameters

in	*device*	Device context pointer
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.37 calib_ecdh_tempkey_ioenc()

```
ATCA_STATUS calib_ecdh_tempkey_ioenc (
            ATCADevice device,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*device*	Device context pointer
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.38 calib_gendig()

```
ATCA_STATUS calib_gendig (
            ATCADevice device,
            uint8_t zone,
            uint16_t key_id,
            const uint8_t * other_data,
            uint8_t other_data_size )
```

Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.

Parameters

in	*device*	Device context pointer
in	*zone*	Designates the source of the data to hash with TempKey.
in	*key_id*	Indicates the key, OTP block, or message order for shared nonce mode.
in	*other_data*	Four bytes of data for SHA calculation when using a NoMac key, 32 bytes for "Shared Nonce" mode, otherwise ignored (can be NULL).
in	*other_data_size*	Size of other_data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.39 calib_genkey()

```
ATCA_STATUS calib_genkey (
            ATCADevice device,
            uint16_t key_id,
            uint8_t * public_key )
```

Issues GenKey command, which generates a new random private key in slot and returns the public key.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot number where an ECC private key is configured. Can also be ATCA_TEMPKEY_KEYID to generate a private key in TempKey.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.40 calib_genkey_base()

```
ATCA_STATUS calib_genkey_base (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * other_data,
            uint8_t * public_key )
```

Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines what operations the GenKey command performs.
in	*key_id*	Slot to perform the GenKey command on.
in	*other_data*	OtherData for PubKey digest calculation. Can be set to NULL otherwise.
out	*public_key*	If the mode indicates a public key will be calculated, it will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.41 calib_genkey_mac()

```
ATCA_STATUS calib_genkey_mac (
            ATCADevice device,
            uint8_t * public_key,
            uint8_t * mac )
```

Uses Genkey command to calculate SHA256 digest MAC of combining public key and session key.

Parameters

in	*device*	Device Context pointer
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
out	*mac*	Combine public key referenced by keyID with current value of session key, calculate a SHA256 digest and return that MAC here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.42 calib_get_addr()

```
ATCA_STATUS calib_get_addr (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            uint16_t * addr )
```

Compute the address given the zone, slot, block, and offset.

Parameters

in	*zone*	Zone to get address from. Config(0), OTP(1), or Data(2) which requires a slot.
in	*slot*	Slot Id number for data zone and zero for other zones.
in	*block*	Block number within the data or configuration or OTP zone .
in	*offset*	Offset Number within the block of data or configuration or OTP zone.
out	*addr*	Pointer to the address of data or configuration or OTP zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.43 calib_get_pubkey()

```
ATCA_STATUS calib_get_pubkey (
            ATCADevice device,
            uint16_t key_id,
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from an existing private key in a slot.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot number of the private key.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.44 calib_get_zone_size()

```
ATCA_STATUS calib_get_zone_size (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            size_t * size )
```

Gets the size of the specified zone in bytes.

Parameters

in	*device*	Device context pointer
in	*zone*	Zone to get size information from. Config(0), OTP(1), or Data(2) which requires a slot.
in	*slot*	If zone is Data(2), the slot to query for size.
out	*size*	Zone size is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.45 calib_hmac()

```
ATCA_STATUS calib_hmac (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            uint8_t * digest )
```

Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*device*	Device context pointer
in	*mode*	Controls which fields within the device are used in the message.
in	*key_id*	Which key is to be used to generate the response. Bits 0:3 only are used to select a slot but all 16 bits are used in the HMAC message.
out	*digest*	HMAC digest is returned in this buffer (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.46 calib_hw_sha2_256()

```
ATCA_STATUS calib_hw_sha2_256 (
            ATCADevice device,
            const uint8_t * data,
            size_t data_size,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*device*	Device context pointer
in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.47 calib_hw_sha2_256_finish()

```
ATCA_STATUS calib_hw_sha2_256_finish (
            ATCADevice device,
            atca_sha256_ctx_t * ctx,
            uint8_t * digest )
```

Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*device*	Device context pointer
in	*ctx*	SHA256 context
out	*digest*	SHA256 digest is returned here (32 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.48 calib_hw_sha2_256_init()

```
ATCA_STATUS calib_hw_sha2_256_init (
            ATCADevice device,
            atca_sha256_ctx_t * ctx )
```

Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.

Parameters

in	*device*	Device context pointer
in	*ctx*	SHA256 context

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.49 calib_hw_sha2_256_update()

```
ATCA_STATUS calib_hw_sha2_256_update (
            ATCADevice device,
            atca_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add message data to a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*device*	Device context pointer
in	*ctx*	SHA256 context
in	*data*	Message data to be added to hash.
in	*data_size*	Size of data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.50 calib_idle()

```
ATCA_STATUS calib_idle (
            ATCADevice device )
```

idle the CryptoAuth device

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.51 calib_info()

```
ATCA_STATUS calib_info (
            ATCADevice device,
            uint8_t * revision )
```

Use the Info command to get the device revision (DevRev).

Parameters

in	*device*	Device context pointer
out	*revision*	Device revision is returned here (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.52 calib_info_base()

```
ATCA_STATUS calib_info_base (
            ATCADevice device,
            uint8_t mode,
            uint16_t param2,
            uint8_t * out_data )
```

Issues an Info command, which return internal device information and can control GPIO and the persistent latch.

Parameters

in	*device*	Device context pointer
in	*mode*	Selects which mode to be used for info command.
in	*param2*	Selects the particular fields for the mode.
out	*out_data*	Response from info command (4 bytes). Can be set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.53 calib_info_get_latch()

```
ATCA_STATUS calib_info_get_latch (
            ATCADevice device,
            bool * state )
```

Use the Info command to get the persistent latch current state for an ATECC608 device.

Parameters

in	*device*	Device context pointer
out	*state*	The state is returned here. Set (true) or Cler (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.54 calib_info_lock_status()

```
ATCA_STATUS calib_info_lock_status (
            ATCADevice device,
            uint16_t param2,
            uint8_t * is_locked )
```

8.6.3.55 calib_info_privkey_valid()

```
ATCA_STATUS calib_info_privkey_valid (
            ATCADevice device,
            uint16_t key_id,
            uint8_t * is_valid )
```

Use Info command to check ECC Private key stored in key slot is valid or not.

Parameters

in	*device*	Device context pointer
in	*key_id*	ECC private key slot id For ECC204, key_id is 0x00
out	*is_valid*	return private key is valid or invalid

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.56 calib_info_set_latch()

```
ATCA_STATUS calib_info_set_latch (
            ATCADevice device,
            bool state )
```

Use the Info command to set the persistent latch state for an ATECC608 device.

Parameters

in	*device*	Device context pointer
out	*state*	Persistent latch state. Set (true) or clear (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.57 calib_is_locked()

```
ATCA_STATUS calib_is_locked (
            ATCADevice device,
            uint8_t zone,
            bool * is_locked )
```

Executes Read command, which reads the configuration zone to see if the specified zone is locked.

Parameters

in	*device*	Device context pointer
in	*zone*	The zone to query for locked (use LOCK_ZONE_CONFIG or LOCK_ZONE_DATA).
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.58 calib_is_private()

```
ATCA_STATUS calib_is_private (
            ATCADevice device,
```

```
            uint16_t slot,
            bool * is_private )
```

Check if a slot is a private key.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot to query (slot 0-15)
out	*is_private*	return true if private

Returns

ATCA_SUCCESS on success, otherwise an error code

8.6.3.59 calib_is_slot_locked()

```
ATCA_STATUS calib_is_slot_locked (
            ATCADevice device,
            uint16_t slot,
            bool * is_locked )
```

Executes Read command, which reads the configuration zone to see if the specified slot is locked.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot to query for locked (slot 0-15)
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.60 calib_kdf()

```
ATCA_STATUS calib_kdf (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint32_t details,
            const uint8_t * message,
            uint8_t * out_data,
            uint8_t * out_nonce )
```

Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.

Generally this function combines a source key with an input string and creates a result key/digest/array.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines KDF algorithm (PRF,AES,HKDF), source key location, and target key locations.
in	*key_id*	Source and target key slots if locations are in the EEPROM. Source key slot is the LSB and target key slot is the MSB.
in	*details*	Further information about the computation, depending on the algorithm (4 bytes).
in	*message*	Input value from system (up to 128 bytes). Actual size of message is 16 bytes for AES algorithm or is encoded in the MSB of the details parameter for other algorithms.
out	*out_data*	Output of the KDF function is returned here. If the result remains in the device, this can be NULL.
out	*out_nonce*	If the output is encrypted, a 32 byte random nonce generated by the device is returned here. If output encryption is not used, this can be NULL.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.61 calib_lock()

```
ATCA_STATUS calib_lock (
            ATCADevice device,
            uint8_t mode,
            uint16_t summary_crc )
```

The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.

Parameters

in	*device*	Device context pointer
in	*mode*	Zone, and/or slot, and summary check (bit 7).
in	*summary_crc*	CRC of the config or data zones. Ignored for slot locks or when mode bit 7 is set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.62 calib_lock_config_zone()

```
ATCA_STATUS calib_lock_config_zone (
            ATCADevice device )
```

Unconditionally (no CRC required) lock the config zone.

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.63 calib_lock_config_zone_crc()

```
ATCA_STATUS calib_lock_config_zone_crc (
            ATCADevice device,
            uint16_t summary_crc )
```

Lock the config zone with summary CRC.

The CRC is calculated over the entire config zone contents. 88 bytes for ATSHA devices, 128 bytes for ATECC devices. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*device*	Device context pointer
in	*summary_crc*	Expected CRC over the config zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.64 calib_lock_data_slot()

```
ATCA_STATUS calib_lock_data_slot (
            ATCADevice device,
            uint16_t slot )
```

Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).

Parameters

in	*device*	Device context pointer
in	*slot*	Slot to be locked in data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.65 calib_lock_data_zone()

```
ATCA_STATUS calib_lock_data_zone (
            ATCADevice device )
```

Unconditionally (no CRC required) lock the data zone (slots and OTP).

ConfigZone must be locked and DataZone must be unlocked for the zone to be successfully locked.

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.66 calib_lock_data_zone_crc()

```
ATCA_STATUS calib_lock_data_zone_crc (
            ATCADevice device,
            uint16_t summary_crc )
```

Lock the data zone (slots and OTP) with summary CRC.

The CRC is calculated over the concatenated contents of all the slots and OTP at the end. Private keys (Key↩Config.Private=1) are skipped. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*device*	Device context pointer
in	*summary_crc*	Expected CRC over the data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.67 calib_mac()

```
ATCA_STATUS calib_mac (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * challenge,
            uint8_t * digest )
```

Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*device*	Device context pointer
in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge message (32 bytes). May be NULL if mode indicates a challenge isn't required.
out	*digest*	MAC response is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.68 calib_nonce()

```
ATCA_STATUS calib_nonce (
            ATCADevice device,
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*device*	Device context pointer
in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.69 calib_nonce_base()

```
ATCA_STATUS calib_nonce_base (
            ATCADevice device,
```

```
            uint8_t mode,
            uint16_t param2,
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.

Parameters

in	*device*	Device context pointer
in	*mode*	Controls the mechanism of the internal RNG or fixed write.
in	*param2*	Param2, normally 0, but can be used to indicate a nonce calculation mode (bit 15). For ECC204, represent tarnsport key id greater than or equal to 0x8000
in	*num_in*	Input value to either be included in the nonce calculation in random modes (20 bytes) or to be written directly (32 bytes or 64 bytes(ATECC608)) in pass-through mode.
out	*rand_out*	If using a random mode, the internally generated 32-byte random number that was used in the nonce calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.70 calib_nonce_gen_session_key()

```
ATCA_STATUS calib_nonce_gen_session_key (
            ATCADevice device,
            uint16_t param2,
            uint8_t * num_in,
            uint8_t * rand_out )
```

Use Nonce command to generate session key for use by a subsequent write command This Mode only supports in ECC204 device.

Parameters

in	*device*	Device context pointer
in	*param2*	Key id points to transport key
in	*num_in*	Input value from host system
out	*rand_out*	Internally generate random number of 32 bytes returned here

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.71 calib_nonce_load()

```
ATCA_STATUS calib_nonce_load (
            ATCADevice device,
            uint8_t target,
            const uint8_t * num_in,
            uint16_t num_in_size )
```

Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.

For the ATECC608, available targets are TempKey (32 or 64 bytes), Message Digest Buffer (32 or 64 bytes), or the Alternate Key Buffer (32 bytes). For all other devices, only TempKey (32 bytes) is available.

Parameters

in	*device*	Device context pointer
in	*target*	Target device buffer to load. Can be NONCE_MODE_TARGET_TEMPKEY, NONCE_MODE_TARGET_MSGDIGBUF, or NONCE_MODE_TARGET_ALTKEYBUF.
in	*num_in*	Data to load into the buffer.
in	*num_in_size*	Size of num_in in bytes. Can be 32 or 64 bytes depending on device and target.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.72 calib_nonce_rand()

```
ATCA_STATUS calib_nonce_rand (
            ATCADevice device,
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.

Parameters

in	*device*	Device context pointer
in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.73 calib_priv_write()

```
ATCA_STATUS calib_priv_write (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t priv_key[36],
            uint16_t write_key_id,
            const uint8_t write_key[32],
            const uint8_t num_in[(20)] )
```

8.6.3.74 calib_random()

```
ATCA_STATUS calib_random (
            ATCADevice device,
            uint8_t * rand_out )
```

Executes Random command, which generates a 32 byte random number from the CryptoAuth device.

Parameters

in	*device*	Device context pointer
out	*rand_out*	32 bytes of random data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.75 calib_read_bytes_zone()

```
ATCA_STATUS calib_read_bytes_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            size_t offset,
            uint8_t * data,
            size_t length )
```

Used to read an arbitrary number of bytes from any zone configured for clear reads.

This function will issue the Read command as many times as is required to read the requested data.

Parameters

in	*device*	Device context pointer
in	*zone*	Zone to read data from. Option are ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	Slot number to read from if zone is ATCA_ZONE_DATA(2). Ignored for all other zones.
in	*offset*	Byte offset within the zone to read from.
out	*data*	Read data is returned here.
in	*length*	Number of bytes to read starting from the offset.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.76 calib_read_config_zone()

```
ATCA_STATUS calib_read_config_zone (
            ATCADevice device,
            uint8_t * config_data )
```

Executes Read command to read the complete device configuration zone.

Parameters

in	*device*	Device context pointer
out	*config_data*	Configuration zone data is returned here. 88 bytes for ATSHA devices, 128 bytes for ATECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.77 calib_read_enc()

```
ATCA_STATUS calib_read_enc (
            ATCADevice device,
            uint16_t key_id,
            uint8_t block,
            uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[(20)] )
```

8.6.3.78 calib_read_pubkey()

```
ATCA_STATUS calib_read_pubkey (
            ATCADevice device,
            uint16_t slot,
            uint8_t * public_key )
```

Executes Read command to read an ECC P256 public key from a slot configured for clear reads.

This function assumes the public key is stored using the ECC public key format specified in the datasheet.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a public key.
out	*public_key*	Public key is returned here (64 bytes). Format will be the 32 byte X and Y big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.79 calib_read_serial_number()

```
ATCA_STATUS calib_read_serial_number (
            ATCADevice device,
            uint8_t * serial_number )
```

Executes Read command, which reads the 9 byte serial number of the device from the config zone.

Parameters

in	*device*	Device context pointer
out	*serial_number*	9 byte serial number is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.80 calib_read_sig()

```
ATCA_STATUS calib_read_sig (
            ATCADevice device,
            uint16_t slot,
            uint8_t * sig )
```

Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a signature.
out	*sig*	Signature will be returned here (64 bytes). Format will be the 32 byte R and S big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.81 calib_read_zone()

```
ATCA_STATUS calib_read_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            uint8_t * data,
            uint8_t len )
```

Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.

When reading a slot or OTP, data zone must be locked and the slot configuration must not be secret for a slot to be successfully read.

Parameters

in	*device*	Device context pointer
in	*zone*	Zone to be read from device. Options are ATCA_ZONE_CONFIG, ATCA_ZONE_OTP, or ATCA_ZONE_DATA.
in	*slot*	Slot number for data zone and ignored for other zones.
in	*block*	32 byte block index within the zone.
in	*offset*	4 byte work index within the block. Ignored for 32 byte reads.
out	*data*	Read data is returned here.
in	*len*	Length of the data to be read. Must be either 4 or 32.

returns ATCA_SUCCESS on success, otherwise an error code.

8.6.3.82 calib_secureboot()

```
ATCA_STATUS calib_secureboot (
            ATCADevice device,
            uint8_t mode,
            uint16_t param2,
            const uint8_t * digest,
            const uint8_t * signature,
            uint8_t * mac )
```

Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines what operations the SecureBoot command performs.

Parameters

in	*param2*	Not used, must be 0.
in	*digest*	Digest of the code to be verified (32 bytes).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
out	*mac*	Validating MAC will be returned here (32 bytes). Can be NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.83 calib_secureboot_mac()

```
ATCA_STATUS calib_secureboot_mac (
            ATCADevice device,
            uint8_t mode,
            const uint8_t * digest,
            const uint8_t * signature,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines what operations the SecureBoot command performs.
in	*digest*	Digest of the code to be verified (32 bytes). This is the plaintext digest (not encrypted).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
in	*num_in*	Host nonce (20 bytes).
in	*io_key*	IO protection key (32 bytes).
out	*is_verified*	Verify result is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.84 calib_selftest()

```
ATCA_STATUS calib_selftest (
            ATCADevice device,
            uint8_t mode,
```

```
            uint16_t param2,
            uint8_t * result )
```

Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608 chip.

Parameters

in	*device*	Device context pointer
in	*mode*	Functions to test. Can be a bit field combining any of the following: SELFTEST_MODE_RNG, SELFTEST_MODE_ECDSA_VERIFY, SELFTEST_MODE_ECDSA_SIGN, SELFTEST_MODE_ECDH, SELFTEST_MODE_AES, SELFTEST_MODE_SHA, SELFTEST_MODE_ALL.
in	*param2*	Currently unused, should be 0.
out	*result*	Results are returned here as a bit field.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.85 calib_sha()

```
ATCA_STATUS calib_sha (
            ATCADevice device,
            uint16_t length,
            const uint8_t * message,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*device*	Device context pointer
in	*length*	Size of message parameter in bytes.
in	*message*	Message data to be hashed.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.86 calib_sha_base()

```
ATCA_STATUS calib_sha_base (
            ATCADevice device,
```

```
            uint8_t mode,
            uint16_t length,
            const uint8_t * message,
            uint8_t * data_out,
            uint16_t * data_out_size )
```

Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.

Only the Start(0) and Compute(1) modes are available for ATSHA devices.

Parameters

in	*device*	Device context pointer
in	*mode*	SHA command mode Start(0), Update/Compute(1), End(2), Public(3), HMACstart(4), HMACend(5), Read_Context(6), or Write_Context(7). Also message digest target location for the ATECC608.
in	*length*	Number of bytes in the message parameter or KeySlot for the HMAC key if Mode is HMACstart(4) or Public(3).
in	*message*	Message bytes to be hashed or Write_Context if restoring a context on the ATECC608. Can be NULL if not required by the mode.
out	*data_out*	Data returned by the command (digest or context).
in,out	*data_out_size*	As input, the size of the data_out buffer. As output, the number of bytes returned in data_out.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.87 calib_sha_end()

```
ATCA_STATUS calib_sha_end (
            ATCADevice device,
            uint8_t * digest,
            uint16_t length,
            const uint8_t * message )
```

Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
out	*digest*	Digest from SHA-256 or HMAC/SHA-256 will be returned here (32 bytes).
in	*length*	Length of any remaining data to include in hash. Max 64 bytes.
in	*message*	Remaining data to include in hash. NULL if length is 0.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.88 calib_sha_hmac()

```
ATCA_STATUS calib_sha_hmac (
            ATCADevice device,
            const uint8_t * data,
            size_t data_size,
            uint16_t key_slot,
            uint8_t * digest,
            uint8_t target )
```

Use the SHA command to compute an HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
in	*key_slot*	Slot key id to use for the HMAC calculation
out	*digest*	Digest is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.89 calib_sha_hmac_finish()

```
ATCA_STATUS calib_sha_hmac_finish (
            ATCADevice device,
            atca_hmac_sha256_ctx_t * ctx,
            uint8_t * digest,
            uint8_t target )
```

Executes SHA command to complete a HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	HMAC/SHA-256 context
out	*digest*	HMAC/SHA-256 result is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option. For ECC204, target is ignored (0x00)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.90 calib_sha_hmac_init()

```
ATCA_STATUS calib_sha_hmac_init (
            ATCADevice device,
            atca_hmac_sha256_ctx_t * ctx,
            uint16_t key_slot )
```

Executes SHA command to start an HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	HMAC/SHA-256 context
in	*key_slot*	Slot key id to use for the HMAC calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.91 calib_sha_hmac_update()

```
ATCA_STATUS calib_sha_hmac_update (
            ATCADevice device,
            atca_hmac_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	HMAC/SHA-256 context
in	*data*	Message data to add
in	*data_size*	Size of message data in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.92 calib_sha_read_context()

```
ATCA_STATUS calib_sha_read_context (
            ATCADevice device,
            uint8_t * context,
            uint16_t * context_size )
```

Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.

Parameters

in	*device*	Device context pointer
out	*context*	Context data is returned here.
in,out	*context_size*	As input, the size of the context buffer in bytes. As output, the size of the returned context data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.93 calib_sha_start()

```
ATCA_STATUS calib_sha_start (
            ATCADevice device )
```

Executes SHA command to initialize SHA-256 calculation engine.

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.94 calib_sha_update()

```
ATCA_STATUS calib_sha_update (
            ATCADevice device,
            const uint8_t * message )
```

Executes SHA command to add 64 bytes of message data to the current context.

Parameters

in	*device*	Device context pointer
in	*message*	64 bytes of message data to add to add to operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.95 calib_sha_write_context()

```
ATCA_STATUS calib_sha_write_context (
            ATCADevice device,
            const uint8_t * context,
            uint16_t context_size )
```

Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.

Parameters

in	*device*	Device context pointer
in	*context*	Context data to be restored.
in	*context_size*	Size of the context data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.96 calib_sign()

```
ATCA_STATUS calib_sign (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * msg,
            uint8_t * signature )
```

Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of the private key to be used to sign the message.
in	*msg*	32-byte message to be signed. Typically the SHA256 hash of the full message.
out	*signature*	Signature will be returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.97 calib_sign_base()

```
ATCA_STATUS calib_sign_base (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            uint8_t * signature )
```

Executes the Sign command, which generates a signature using the ECDSA algorithm.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines what the source of the message to be signed.
in	*key_id*	Private key slot used to sign the message.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.98 calib_sign_internal()

```
ATCA_STATUS calib_sign_internal (
            ATCADevice device,
            uint16_t key_id,
            bool is_invalidate,
            bool is_full_sn,
            uint8_t * signature )
```

Executes Sign command to sign an internally generated message.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of the private key to be used to sign the message.
in	*is_invalidate*	Set to true if the signature will be used with the Verify(Invalidate) command. false for all other cases.
in	*is_full_sn*	Set to true if the message should incorporate the device's full serial number.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.99 calib_sleep()

```
ATCA_STATUS calib_sleep (
            ATCADevice device )
```

invoke sleep on the CryptoAuth device

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.100 calib_updateextra()

```
ATCA_STATUS calib_updateextra (
            ATCADevice device,
            uint8_t mode,
            uint16_t new_value )
```

Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

Can also be used to decrement the limited use counter associated with the key in slot NewValue.

Parameters

in	*device*	Device context pointer
in	*mode*	Mode determines what operations the UpdateExtra command performs.
in	*new_value*	Value to be written.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.101 calib_verify()

```
ATCA_STATUS calib_verify (
            ATCADevice device,
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * public_key,
            const uint8_t * other_data,
            uint8_t * mac )
```

Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.

For the Stored, External, and ValidateExternal Modes, the contents of TempKey (or Message Digest Buffer in some cases for the ATECC608) should contain the 32 byte message.

Parameters

in	*device*	Device context pointer
in	*mode*	Verify command mode and options
in	*key_id*	Stored mode, the slot containing the public key to be used for the verification. ValidateExternal mode, the slot containing the public key to be validated. External mode, KeyID contains the curve type to be used to Verify the signature. Validate or Invalidate mode, the slot containing the public key to be (in)validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	If mode is External, the public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve. NULL for all other modes.
in	*other_data*	If mode is Validate, the bytes used to generate the message for the validation (19 bytes). NULL for all other modes.
out	*mac*	If mode indicates a validating MAC, then the MAC will will be returned here. Can be NULL otherwise.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.102 calib_verify_extern()

```
ATCA_STATUS calib_verify_extern (
            ATCADevice device,
            const uint8_t * message,
            const uint8_t * signature,
            const uint8_t * public_key,
            bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.103 calib_verify_extern_mac()

```
ATCA_STATUS calib_verify_extern_mac (
            ATCADevice device,
            const uint8_t * message,
            const uint8_t * signature,
            const uint8_t * public_key,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.104 calib_verify_invalidate()

```
ATCA_STATUS calib_verify_invalidate (
            ATCADevice device,
```

```
        uint16_t key_id,
        const uint8_t * signature,
        const uint8_t * other_data,
        bool * is_verified )
```

Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be invalidated in TempKey (mode=0x10).

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot containing the public key to be invalidated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.105 calib_verify_stored()

```
ATCA_STATUS calib_verify_stored (
        ATCADevice device,
        const uint8_t * message,
        const uint8_t * signature,
        uint16_t key_id,
        bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.106 calib_verify_stored_mac()

```
ATCA_STATUS calib_verify_stored_mac (
            ATCADevice device,
            const uint8_t * message,
            const uint8_t * signature,
            uint16_t key_id,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.

Parameters

in	*device*	Device context pointer
in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.107 calib_verify_validate()

```
ATCA_STATUS calib_verify_validate (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * other_data,
            bool * is_verified )
```

Executes the Verify command in Validate mode to validate a public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be validated in TempKey (mode=0x10).

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot containing the public key to be validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

8.6.3.108 calib_wakeup()

```
ATCA_STATUS calib_wakeup (
            ATCADevice device )
```

wakeup the CryptoAuth device

Parameters

in	*device*	Device context pointer

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.109 calib_write()

```
ATCA_STATUS calib_write (
            ATCADevice device,
            uint8_t zone,
            uint16_t address,
            const uint8_t * value,
            const uint8_t * mac )
```

Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.

Parameters

in	*device*	Device context pointer
in	*zone*	Zone/Param1 for the write command.
in	*address*	Address/Param2 for the write command.
in	*value*	Plain-text data to be written or cipher-text for encrypted writes. 32 or 4 bytes depending on bit 7 in the zone.
in	*mac*	MAC required for encrypted writes (32 bytes). Set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.110 calib_write_bytes_zone()

```
ATCA_STATUS calib_write_bytes_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            size_t offset_bytes,
            const uint8_t * data,
            size_t length )
```

Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).

Config zone must be unlocked for writes to that zone. If data zone is unlocked, only 32-byte writes are allowed to slots and OTP and the offset and length must be multiples of 32 or the write will fail.

Parameters

in	*device*	Device context pointer
in	*zone*	Zone to write data to: ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	If zone is ATCA_ZONE_DATA(2), the slot number to write to. Ignored for all other zones.
in	*offset_bytes*	Byte offset within the zone to write to. Must be a multiple of a word (4 bytes).
in	*data*	Data to be written.
in	*length*	Number of bytes to be written. Must be a multiple of a word (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.111 calib_write_config_counter()

```
ATCA_STATUS calib_write_config_counter (
            ATCADevice device,
            uint16_t counter_id,
            uint32_t counter_value )
```

Initialize one of the monotonic counters in device with a specific value.

The monotonic counters are stored in the configuration zone using a special format. This encodes a binary count value into the 8 byte encoded value required. Can only be set while the configuration zone is unlocked.

Parameters

in	*device*	Device context pointer
in	*counter_id*	Counter to be written.
in	*counter_value*	Counter value to set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.112 calib_write_config_zone()

```
ATCA_STATUS calib_write_config_zone (
            ATCADevice device,
            const uint8_t * config_data )
```

Executes the Write command, which writes the configuration zone.

First 16 bytes are skipped as they are not writable. LockValue and LockConfig are also skipped and can only be changed via the Lock command.

This command may fail if UserExtra and/or Selector bytes have already been set to non-zero values.

Parameters

in	*device*	Device context pointer
in	*config_data*	Data to the config zone data. This should be 88 bytes for SHA devices and 128 bytes for ECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.113 calib_write_enc()

```
ATCA_STATUS calib_write_enc (
            ATCADevice device,
            uint16_t key_id,
            uint8_t block,
            const uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[(20)] )
```

8.6.3.114 calib_write_pubkey()

```
ATCA_STATUS calib_write_pubkey (
            ATCADevice device,
            uint16_t slot,
            const uint8_t * public_key )
```

Uses the write command to write a public key to a slot in the proper format.

Parameters

in	*device*	Device context pointer
in	*slot*	Slot number to write. Only slots 8 to 15 are large enough to store a public key.
in	*public_key*	Public key to write into the slot specified. X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.3.115 calib_write_zone()

```
ATCA_STATUS calib_write_zone (
            ATCADevice device,
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            const uint8_t * data,
            uint8_t len )
```

Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.

Parameters

in	*device*	Device context pointer
in	*zone*	Device zone to write to (0=config, 1=OTP, 2=data).
in	*slot*	If writing to the data zone, it is the slot to write to, otherwise it should be 0.
in	*block*	32-byte block to write to.
in	*offset*	4-byte word within the specified block to write to. If performing a 32-byte write, this should be 0.
in	*data*	Data to be written.
in	*len*	Number of bytes to be written. Must be either 4 or 32.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.6.4 Variable Documentation

8.6.4.1 atca_basic_aes_gcm_version

```
const char* atca_basic_aes_gcm_version = "2.0"
```

8.7 Software crypto methods (atcac_)

These methods provide a software implementation of various crypto algorithms.

8.7.0.1 crypto directory - Purpose

This directory contains software implementations of cryptographic functions. The functions at the base level are wrappers that will point to the final implementations of the software crypto functions.

Macros

- #define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
- #define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
- #define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE ∗ 2)
- #define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE ∗ 2)

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[(256/8)], const uint8_t signature[((256/8) ∗2)], const uint8_t public_key[((256/8) ∗2)])

    return software generated ECDSA verification result and the function is currently not implemented
- int atcac_sw_random (uint8_t ∗data, size_t data_size)

    Return Random Bytes.
- int atcac_sw_sha1_init (atcac_sha1_ctx ∗ctx)

    Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

    Add data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx ∗ctx, uint8_t digest[(20)])
- int atcac_sw_sha1 (const uint8_t ∗data, size_t data_size, uint8_t digest[(20)])

    Perform SHA1 hash of data in software.
- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx ∗ctx)

    Initialize context for performing SHA256 hash in software.
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

    Add data to a SHA256 hash.
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx ∗ctx, uint8_t digest[(32)])
- int atcac_sw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t digest[(32)])

    single call convenience function which computes Hash of given data using SHA256 software
- ATCA_STATUS atcac_sha256_hmac_init (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len)

    Initialize context for performing HMAC (sha256) in software.
- ATCA_STATUS atcac_sha256_hmac_update (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

    Update HMAC context with input data.
- ATCA_STATUS atcac_sha256_hmac_finish (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗digest, size_t ∗digest_len)

    Finish CMAC calculation and clear the HMAC context.
- ATCA_STATUS atcac_sha256_hmac_counter (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗label, size_t label_len, uint8_t ∗data, size_t data_len, uint8_t ∗digest, size_t diglen)

    Implements SHA256 HMAC-Counter per NIST SP 800-108 used for KDF like operations.

8.7.1 Detailed Description

These methods provide a software implementation of various crypto algorithms.

8.7.2 Macro Definition Documentation

8.7.2.1 ATCA_ECC_P256_FIELD_SIZE

```
#define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
```

8.7.2.2 ATCA_ECC_P256_PRIVATE_KEY_SIZE

```
#define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
```

8.7.2.3 ATCA_ECC_P256_PUBLIC_KEY_SIZE

```
#define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
```

8.7.2.4 ATCA_ECC_P256_SIGNATURE_SIZE

```
#define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
```

8.7.3 Function Documentation

8.7.3.1 atcac_sha256_hmac_counter()

```
ATCA_STATUS atcac_sha256_hmac_counter (
            atcac_hmac_sha256_ctx * ctx,
            uint8_t * label,
            size_t label_len,
            uint8_t * data,
            size_t data_len,
            uint8_t * digest,
            size_t diglen )
```

Implements SHA256 HMAC-Counter per NIST SP 800-108 used for KDF like operations.

8.7.3.2 atcac_sha256_hmac_finish()

```
ATCA_STATUS atcac_sha256_hmac_finish (
            atcac_hmac_sha256_ctx * ctx,
            uint8_t * digest,
            size_t * digest_len )
```

Finish CMAC calculation and clear the HMAC context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a sha256-hmac context
out	*digest*	hmac value
in,out	*digest_len*	length of hmac

8.7.3.3 atcac_sha256_hmac_init()

```
ATCA_STATUS atcac_sha256_hmac_init (
            atcac_hmac_sha256_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len )
```

Initialize context for performing HMAC (sha256) in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a sha256-hmac context
in	*key*	key value to use
in	*key_len*	length of the key

8.7.3.4 atcac_sha256_hmac_update()

```
ATCA_STATUS atcac_sha256_hmac_update (
            atcac_hmac_sha256_ctx * ctx,
            const uint8_t * data,
            size_t data_size )
```

Update HMAC context with input data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a sha256-hmac context
in	*data*	input data
in	*data_size*	length of input data

8.7.3.5 atcac_sw_ecdsa_verify_p256()

```
int atcac_sw_ecdsa_verify_p256 (
            const uint8_t msg[(256/8)],
            const uint8_t signature[((256/8) *2)],
            const uint8_t public_key[((256/8) *2)] )
```

return software generated ECDSA verification result and the function is currently not implemented

Parameters

in	*msg*	ptr to message or challenge
in	*signature*	ptr to the signature to verify
in	*public_key*	ptr to public key of device which signed the challenge return ATCA_UNIMPLEMENTED , as the function is currently not implemented

8.7.3.6 atcac_sw_random()

```
int atcac_sw_random (
            uint8_t * data,
            size_t data_size )
```

Return Random Bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.7.3.7 atcac_sw_sha1()

```
int atcac_sw_sha1 (
            const uint8_t * data,
            size_t data_size,
            uint8_t digest[(20)] )
```

Perform SHA1 hash of data in software.

Parameters

in	*data*	Data to be hashed
in	*data_size*	Data size in bytes
out	*digest*	Digest is returned here (20 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.7.3.8 atcac_sw_sha1_finish()

```
int atcac_sw_sha1_finish (
            atcac_sha1_ctx * ctx,
            uint8_t digest[(20)] )
```

8.7.3.9 atcac_sw_sha1_init()

```
int atcac_sw_sha1_init (
            atcac_sha1_ctx * ctx )
```

Initialize context for performing SHA1 hash in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context

8.7.3.10 atcac_sw_sha1_update()

```
int atcac_sw_sha1_update (
            atcac_sha1_ctx * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add data to a SHA1 hash.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
in	*data*	input data buffer
in	*data_size*	input data length

8.7.3.11 atcac_sw_sha2_256()

```
int atcac_sw_sha2_256 (
            const uint8_t * data,
            size_t data_size,
            uint8_t digest[(32)] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*data*	pointer to stream of data to hash
in	*data_size*	size of data stream to hash
out	*digest*	result

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.7.3.12 atcac_sw_sha2_256_finish()

```
int atcac_sw_sha2_256_finish (
            atcac_sha2_256_ctx * ctx,
            uint8_t digest[(32)] )
```

8.7.3.13 atcac_sw_sha2_256_init()

```
int atcac_sw_sha2_256_init (
            atcac_sha2_256_ctx * ctx )
```

Initialize context for performing SHA256 hash in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context

8.7.3.14 atcac_sw_sha2_256_update()

```
int atcac_sw_sha2_256_update (
            atcac_sha2_256_ctx * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add data to a SHA256 hash.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
in	*data*	input data buffer
in	*data_size*	input data length

8.8 Hardware abstraction layer (hal_)

These methods define the hardware abstraction layer for communicating with a CryptoAuth device.

8.8.0.1 HAL Directory - Purpose

This directory contains all the Hardware Abstraction Layer (HAL) files used to adapt the upper levels of atca-ng and abstractions to physical hardware.

HAL contains physical implementations for I2C, SWI, SPI, UART and timers for specific hardware platforms.

Include just those HAL files you require based on platform type.

Cryptoauthlib HAL Architecture

Cryptoauthlib has several intermediate conceptual layers

1. The highest layer of cryptoauthlib (outside of integration APIS) that may be used with an application is the atcab_ api functions. These are general purpose functions that present a simple and consistent crypto interface to the application regardless of the device being used.
2. calib_, talib_ APIs are the library functions behind atcab_ ones that generate the correct command packets and process the received responses. Device specific logic is handled by the library here
3. hal_ these functions perform the transmit/recieve of data for a given interface. These are split into sublayers
   - The HAL layer is the first hal layer that presents the interface expected by the higher level library. When using a native driver and no further interpretation is required this layer is all that is required.
   - The PHY layer if for hals that perform an interpretation or additional protocol logic. In this situation the HAL performs protocol interpretation while the phy performs the physical communication

HAL and PHY Requirements The hal and phy layers have the same construction. A hal or phy must have the following functions and their signatures

- ATCA_STATUS hal_<name>*init(ATCAIface iface, ATCAIfaceCfg *cfg);*
- *ATCA_STATUS hal*<name>*post_init(ATCAIface iface);*
- *ATCA_STATUS hal*<name>*send(ATCAIface iface, uint8_t address, uint8_t *txdata, int txlength);*
- *ATCA_STATUS hal*<name>*receive(ATCAIface iface, uint8_t address, uint8_t *rxdata, uint16_t *rxlength);*
- *ATCA_STATUS hal*<name>*control(ATCAIface iface, uint8_t option, void* param, size_t paramlen);*
- *ATCA_STATUS hal*<name>_release(void *hal_data);

If the hal is a native driver no phy is required. See the tables below for which hal is required to be ported based on a configured interface

CryptoAuthLib Supported HAL Layers

Device Interface	Physical Interface	HAL	PHY
i2c	i2c	hal_i2c	
	gpio	hal_i2c_gpio	hal_gpio
spi	spi	hal_spi	
swi	uart	hal_swi	hal_uart
	gpio	hal_swi_gpio	hal_gpio
any	uart	kit	hal_uart
	hid	kit	hal_hid
	any (user provided)	kit_bridge	

Microchip Harmony 3 for all PIC32 & ARM products - Use the Harmony 3 Configurator to generate and configure prjects

Obtain library and configure using `Harmony 3`

Interface	Files	API	Notes
I2C	hal_i2c_harmony.c	plib.h	For all Harmony 3 based projects
SPI	hal_spi_harmony.c	plib.h	
UART	hal_uart_harmony.c	plib.h	}

Microchip 8 & 16 bit products - AVR, PIC16/18, PIC24/DSPIC

Obtain library and integration through `Microchip Code Configurator`

OS & RTOS integrations

Use `CMake` to configure the library in Linux, Windows, and MacOS environments

OS	Interface	Files	API	Notes
Linux	I2C	hal_linux_i2c_userspace.c/h	i2c-dev	
Linux	SPI	hal_linux_spi_userspace.c/h	spidev	
Linux/Mac		hal_linux.c		For all Linux/Mac projects
Windows		hal_windows.c		For all Windows projects
All	kit-hid	hal_all_platforms_kit_hidapi.c/h	hidapi	Works for Windows, Linux, and Mac
freeRTOS		hal_freertos.c		freeRTOS common routines

Legacy Support - `Atmel START` for AVR, ARM based processesors (SAM)

Interface	Files	API	Notes
	hal_timer_start.c	START	Timer implementation
I2C	hal_i2c_start.c/h	START	
SWI	swi_uart_start.c/h	START	SWI using UART

Legacy Support - ASF3 for ARM Cortex-m0 & Cortex-m based processors (SAM)

SAM Micros	Interface	Files	API	Notes
cortex-m0	I2C	hal_sam0_i2c_asf.c/h	ASF3	SAMD21, SAMB11, etc
cortex-m3/4/7	I2C	hal_sam_i2c_asf.c/h	ASF3	SAM4S, SAMG55, SAMV71, etc
all		hal_sam_timer_asf.c	ASF3	Common timer hal for all platforms

Data Structures

- struct atca_hal_kit_phy_t
- struct i2c_start_instance
- struct atca_i2c_host_s
- struct i2c_sam_instance
- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL for ASF SERCOM
- struct atcaSWImaster

  this is the hal_data for ATCA HAL for ASF SERCOM

Macros

- #define ATCA_POLLING_INIT_TIME_MSEC 1
- #define ATCA_POLLING_FREQUENCY_TIME_MSEC 2
- #define ATCA_POLLING_MAX_TIME_MSEC 2500
- #define hal_memset_s atcab_memset_s
- #define MAX_I2C_BUSES 3
- #define KIT_MAX_SCAN_COUNT 4
- #define KIT_MAX_TX_BUF 32
- #define KIT_TX_WRAP_SIZE (10)
- #define KIT_MSG_SIZE (32)
- #define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)
- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90
- #define DEBUG_PIN_1 EXT2_PIN_5
- #define DEBUG_PIN_2 EXT2_PIN_6
- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 93

Typedefs

- typedef void(∗ start_change_baudrate) (ATCAIface iface, uint32_t speed)
- typedef struct i2c_start_instance i2c_start_instance_t
- typedef struct atca_i2c_host_s atca_i2c_host_t
- typedef void(∗ sam_change_baudrate) (ATCAIface iface, uint32_t speed)
- typedef struct i2c_sam_instance i2c_sam_instance_t
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Enumerations

- enum ATCA_HAL_CONTROL {
ATCA_HAL_CONTROL_WAKE = 0, ATCA_HAL_CONTROL_IDLE = 1, ATCA_HAL_CONTROL_SLEEP = 2, ATCA_HAL_CONTROL_RESET = 3,
ATCA_HAL_CONTROL_SELECT = 4, ATCA_HAL_CONTROL_DESELECT = 5, ATCA_HAL_CHANGE_BAUD = 6, ATCA_HAL_FLUSH_BUFFER = 7 }

Functions

- ATCA_STATUS hal_iface_init (ATCAIfaceCfg ∗, ATCAHAL_t ∗∗hal, ATCAHAL_t ∗∗phy)

    Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType, void ∗hal_data)

    releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t ∗response, int response_size)

    Utility function for hal_wake to check the reply.
- void atca_delay_ms (uint32_t ms)

    Timer API for legacy implementations.
- void atca_delay_us (uint32_t delay)

    This function delays for a number of microseconds.
- void hal_rtos_delay_ms (uint32_t ms)

    Timer API implemented at the HAL level.
- void hal_delay_ms (uint32_t delay)

    This function delays for a number of milliseconds.
- void hal_delay_us (uint32_t delay)

    This function delays for a number of microseconds.
- ATCA_STATUS hal_create_mutex (void ∗∗ppMutex, char ∗pName)

    Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void ∗pMutex)
- ATCA_STATUS hal_lock_mutex (void ∗pMutex)
- ATCA_STATUS hal_unlock_mutex (void ∗pMutex)
- void ∗ hal_malloc (size_t size)
- void hal_free (void ∗ptr)
- ATCA_STATUS hal_iface_register_hal (ATCAIfaceType iface_type, ATCAHAL_t ∗hal, ATCAHAL_t ∗∗old←_hal, ATCAHAL_t ∗phy, ATCAHAL_t ∗∗old_phy)

    Register/Replace a HAL with a.
- uint8_t hal_is_command_word (uint8_t word_address)

    Utility function for hal_wake to check the reply.
- ATCA_STATUS hal_kit_hid_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

    HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

    HAL implementation of Kit HID post init.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

    HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_←t ∗rxsize)

    HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

    Perform control operations for the kit protocol.
- ATCA_STATUS hal_kit_hid_release (void ∗hal_data)

    Close the physical port for HID.

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speec of I2C
- ATCA_STATUS hal_i2c_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_kit_attach_phy (ATCAIfaceCfg ∗cfg, atca_hal_kit_phy_t ∗phy)

  Helper function that connects a physical layer context structure that will be used by the kit protocol bridge.
- ATCA_STATUS hal_kit_discover_buses (int busses[ ], int max_buses)

  Request a list of busses from the kit host.
- ATCA_STATUS hal_kit_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_post_init (ATCAIface iface)

  HAL implementation of Kit HID post init.
- ATCA_STATUS hal_kit_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_control (ATCAIface iface, uint8_t option)

  Kit Protocol Control.
- ATCA_STATUS hal_kit_release (void ∗hal_data)

  Close the physical port for HID.
- void hal_delay_10us (uint32_t delay)

This function delays for a number of tens of microseconds.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- ATCA_STATUS hal_spi_discover_buses (int spi_buses[ ], int max_buses)

  discover spi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_spi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any TA100 devices on a given logical bus number
- ATCA_STATUS hal_spi_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  initialize an SPI interface using given config
- ATCA_STATUS hal_spi_post_init (ATCAIface iface)

  HAL implementation of SPI post init.
- ATCA_STATUS hal_spi_select (ATCAIface iface)

  HAL implementation to assert the device chip select.
- ATCA_STATUS hal_spi_deselect (ATCAIface iface)

  HAL implementation to deassert the device chip select.
- ATCA_STATUS hal_spi_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  HAL implementation of SPI send over Harmony.
- ATCA_STATUS hal_spi_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of SPI receive function for HARMONY SPI.
- ATCA_STATUS hal_spi_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_spi_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist
- ATCA_STATUS hal_swi_discover_buses (int swi_buses[ ], int max_buses)

  discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge
- ATCA_STATUS hal_swi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.
- ATCA_STATUS hal_swi_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

  HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_←
t ∗rxlength)

  Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

  Send Wake flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

  Send Idle flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

  Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_release (void ∗hal_data)

  Manages reference count on given bus and releases resource if no more reference(s) exist.
- ATCA_STATUS hal_swi_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  initialize an SWI interface using given config
- ATCA_STATUS hal_swi_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

Perform control operations for the kit protocol.

- char ∗ strnchr (const char ∗s, size_t count, int c)
- const char ∗ kit_id_from_devtype (ATCADeviceType devtype)
- const char ∗ kit_interface_from_kittype (ATCAKitType kittype)
- ATCA_STATUS kit_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)
- ATCA_STATUS kit_post_init (ATCAIface iface)
- ATCA_STATUS kit_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)
- ATCA_STATUS kit_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxsize)
- ATCA_STATUS kit_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)
- ATCA_STATUS kit_release (void ∗hal_data)
- ATCA_STATUS kit_wrap_cmd (const uint8_t ∗txdata, int txlength, char ∗pkitbuf, int ∗nkitbuf, char target)
- ATCA_STATUS kit_parse_rsp (const char ∗pkitbuf, int nkitbuf, uint8_t ∗kitstatus, uint8_t ∗rxdata, int ∗nrxdata)
- ATCA_STATUS kit_wake (ATCAIface iface)
- ATCA_STATUS kit_idle (ATCAIface iface)
- ATCA_STATUS kit_sleep (ATCAIface iface)
- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Variables

- struct port_config pin_conf

8.8.1 Detailed Description

These methods define the hardware abstraction layer for communicating with a CryptoAuth device.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using SWI Interface.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using SWI bit banging.

These methods define the hardware abstraction layer for communicating with a TA100 device.

< Uncomment when debugging

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using I2C driver of ASF.

8.8.2 Macro Definition Documentation

8.8.2.1 ATCA_POLLING_FREQUENCY_TIME_MSEC

```
#define ATCA_POLLING_FREQUENCY_TIME_MSEC 2
```

8.8.2.2 ATCA_POLLING_INIT_TIME_MSEC

```
#define ATCA_POLLING_INIT_TIME_MSEC 1
```

8.8.2.3 ATCA_POLLING_MAX_TIME_MSEC

```
#define ATCA_POLLING_MAX_TIME_MSEC 2500
```

8.8.2.4 DEBUG_PIN_1

```
#define DEBUG_PIN_1 EXT2_PIN_5
```

8.8.2.5 DEBUG_PIN_2

```
#define DEBUG_PIN_2 EXT2_PIN_6
```

8.8.2.6 hal_memset_s

```
#define hal_memset_s atcab_memset_s
```

8.8.2.7 KIT_MAX_SCAN_COUNT

```
#define KIT_MAX_SCAN_COUNT 4
```

8.8.2.8 KIT_MAX_TX_BUF

```
#define KIT_MAX_TX_BUF 32
```

8.8.2.9 KIT_MSG_SIZE

```
#define KIT_MSG_SIZE (32)
```

8.8.2.10 KIT_RX_WRAP_SIZE

```
#define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)
```

8.8.2.11 KIT_TX_WRAP_SIZE

```
#define KIT_TX_WRAP_SIZE (10)
```

8.8.2.12 MAX_I2C_BUSES

```
#define MAX_I2C_BUSES 3
```

8.8.2.13 MAX_SWI_BUSES [1/2]

```
#define MAX_SWI_BUSES 6
```

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

8.8.2.14 MAX_SWI_BUSES [2/2]

```
#define MAX_SWI_BUSES 6
```

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

8.8.2.15 RECEIVE_MODE [1/2]

```
#define RECEIVE_MODE 0
```

8.8.2.16 RECEIVE_MODE [2/2]

```
#define RECEIVE_MODE 0
```

8.8.2.17 RX_DELAY [1/2]

```
#define RX_DELAY 10
```

8.8.2.18 RX_DELAY [2/2]

```
#define RX_DELAY 10
```

8.8.2.19 TRANSMIT_MODE [1/2]

```
#define TRANSMIT_MODE 1
```

8.8.2.20 TRANSMIT_MODE [2/2]

```
#define TRANSMIT_MODE 1
```

8.8.2.21 TX_DELAY [1/2]

```
#define TX_DELAY 90
```

8.8.2.22 TX_DELAY [2/2]

```
#define TX_DELAY 93
```

8.8.3 Typedef Documentation

8.8.3.1 atca_i2c_host_t

```
typedef struct atca_i2c_host_s atca_i2c_host_t
```

8.8.3.2 ATCAI2CMaster_t

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

8.8.3.3 ATCASWIMaster_t [1/2]

```
typedef struct atcaSWImaster ATCASWIMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

8.8.3.4 ATCASWIMaster_t [2/2]

```
typedef struct atcaSWImaster ATCASWIMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

8.8.3.5 i2c_sam_instance_t

```
typedef struct i2c_sam_instance i2c_sam_instance_t
```

8.8.3.6 i2c_start_instance_t

```
typedef struct i2c_start_instance i2c_start_instance_t
```

8.8.3.7 sam_change_baudrate

```
typedef void(* sam_change_baudrate) (ATCAIface iface, uint32_t speed)
```

8.8.3.8 start_change_baudrate

```
typedef void(* start_change_baudrate) (ATCAIface iface, uint32_t speed)
```

8.8.4 Enumeration Type Documentation

8.8.4.1 ATCA_HAL_CONTROL

```
enum ATCA_HAL_CONTROL
```

Enumerator

ATCA_HAL_CONTROL_WAKE	
ATCA_HAL_CONTROL_IDLE	
ATCA_HAL_CONTROL_SLEEP	
ATCA_HAL_CONTROL_RESET	
ATCA_HAL_CONTROL_SELECT	
ATCA_HAL_CONTROL_DESELECT	
ATCA_HAL_CHANGE_BAUD	
ATCA_HAL_FLUSH_BUFFER	

8.8.5 Function Documentation

8.8.5.1 atca_delay_10us()

```
void atca_delay_10us (
            uint32_t delay )
```

This function delays for a number of tens of microseconds.

Parameters

in	*delay*	number of 0.01 milliseconds to delay

8.8.5.2 atca_delay_ms()

```
void atca_delay_ms (
            uint32_t delay )
```

Timer API for legacy implementations.

This function delays for a number of milliseconds.

```
You can override this function if you like to do
something else in your system while delaying.
```

Parameters

in	*delay*	number of milliseconds to delay

8.8.5.3 atca_delay_us()

```
void atca_delay_us (
            uint32_t delay )
```

This function delays for a number of microseconds.

Parameters

in	*delay*	number of 0.001 milliseconds to delay
in	*delay*	number of microseconds to delay

8.8.5.4 change_i2c_speed()

```
ATCA_STATUS change_i2c_speed (
            ATCAIface iface,
            uint32_t speed )
```

method to change the bus speec of I2C

method to change the bus speed of I2C

Parameters

in	*iface*	interface on which to change bus speed
in	*speed*	baud rate (typically 100000 or 400000)
in	*iface*	interface on which to change bus speed
in	*speed*	baud rate (typically 100000 or 400000)

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.5 hal_check_wake()

```
ATCA_STATUS hal_check_wake (
            const uint8_t * response,
            int response_size )
```

Utility function for hal_wake to check the reply.

Parameters

in	*response*	Wake response to be checked.
in	*response_size*	Size of the response to check.

Returns

ATCA_SUCCESS for expected wake, ATCA_STATUS_SELFTEST_ERROR if the power on self test failed, ATCA_WAKE_FAILED for other failures.

8.8.5.6 hal_create_mutex()

```
ATCA_STATUS hal_create_mutex (
            void ** ppMutex,
            char * pName )
```

Optional hal interfaces.

Application callback for creating a mutex object.

Parameters

in,out	*ppMutex*	location to receive ptr to mutex
in,out	*pName*	String used to identify the mutex
	[IN/OUT]	ppMutex location to receive ptr to mutex
	[IN]	pName Name of the mutex for systems using named objects

8.8.5.7 hal_delay_10us()

```
void hal_delay_10us (
            uint32_t delay )
```

This function delays for a number of tens of microseconds.

Parameters

in	*delay*	number of 0.01 milliseconds to delay

8.8.5.8 hal_delay_ms()

```
void hal_delay_ms (
            uint32_t delay )
```

This function delays for a number of milliseconds.

```
You can override this function if you like to do
something else in your system while delaying.
```

Parameters

in	*delay*	number of milliseconds to delay

8.8.5.9 hal_delay_us()

```
void hal_delay_us (
            uint32_t delay )
```

This function delays for a number of microseconds.

Parameters

in	*delay*	number of microseconds to delay

8.8.5.10 hal_destroy_mutex()

```
ATCA_STATUS hal_destroy_mutex (
            void * pMutex )
```

8.8.5.11 hal_free()

```
void hal_free (
            void * ptr )
```

8.8.5.12 hal_i2c_control()

```
ATCA_STATUS hal_i2c_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.13 hal_i2c_discover_buses()

```
ATCA_STATUS hal_i2c_discover_buses (
            int i2c_buses[],
            int max_buses )
```

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge

This HAL implementation assumes you've included the ASF TWI libraries in your project, otherwise, the HAL layer will not compile because the ASF TWI drivers are a dependency.

logical to physical bus mapping structure

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover return ATCA_SUCCESS

8.8.5.14 hal_i2c_discover_devices()

```
ATCA_STATUS hal_i2c_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	logical bus number on which to look for CryptoAuth devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_SUCCESS

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_SUCCESS

Parameters

in	*bus_num*	Logical bus number on which to look for CryptoAuth devices
out	*cfg*	Pointer to head of an array of interface config structures which get filled in by this method
out	*found*	Number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.15 hal_i2c_idle()

```
ATCA_STATUS hal_i2c_idle (
            ATCAIface iface )
```

idle CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.16 hal_i2c_init() [1/2]

```
ATCA_STATUS hal_i2c_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c↩_init manages these things and ATCAIFace is abstracted from the physical details.

HAL implementation of I2C init.

- this HAL implementation assumes you've included the START Twi libraries in your project, otherwise, the HAL layer will not compile because the START TWI drivers are a dependency ∗

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

this implementation assumes I2C peripheral has been enabled by user. It only initialize an I2C interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.17 hal_i2c_init() [2/2]

```
ATCA_STATUS hal_i2c_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.

hal_i2c_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.

initialize an I2C interface using given config

- this HAL implementation assumes you've included the START Twi libraries in your project, otherwise, the HAL layer will not compile because the START TWI drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the ASF SERCOM I2C libraries in your project, otherwise, the HAL layer will not compile because the ASF I2C drivers are a dependency *

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the ASF Twi libraries in your project, otherwise, the HAL layer will not compile because the ASF TWI drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.18 hal_i2c_post_init()

```
ATCA_STATUS hal_i2c_post_init (
            ATCAIface iface )
```

HAL implementation of I2C post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.19 hal_i2c_receive()

```
ATCA_STATUS hal_i2c_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of I2C receive function for START I2C.

HAL implementation of I2C receive function for ASF I2C.

HAL implementation of I2C receive function.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
in	*address*	device address
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device word address
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.20 hal_i2c_release()

```
ATCA_STATUS hal_i2c_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

manages reference count on given bus and releases resource if no more refernces exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation return ATCA_SUCCESS
in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

8.8.5.21 hal_i2c_send()

```
ATCA_STATUS hal_i2c_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of I2C send over START.

HAL implementation of I2C send over ASF.

HAL implementation of I2C send.

Parameters

in	*iface*	instance
in	*word_address*	device transaction type
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	instance
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	instance
in	*word_address*	device word address
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.22 hal_i2c_sleep()

```
ATCA_STATUS hal_i2c_sleep (
            ATCAIface iface )
```

sleep CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.23 hal_i2c_wake()

```
ATCA_STATUS hal_i2c_wake (
            ATCAIface iface )
```

wake up CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to wakeup

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.24 hal_iface_init()

```
ATCA_STATUS hal_iface_init (
            ATCAIfaceCfg * cfg,
            ATCAHAL_t ** hal,
            ATCAHAL_t ** phy )
```

Standard HAL API for ATCA to initialize a physical interface.

Parameters

in	*cfg*	pointer to ATCAIfaceCfg object
in	*hal*	pointer to ATCAHAL_t intermediate data structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.25 hal_iface_register_hal()

```
ATCA_STATUS hal_iface_register_hal (
            ATCAIfaceType iface_type,
            ATCAHAL_t * hal,
            ATCAHAL_t ** old_hal,
            ATCAHAL_t * phy,
            ATCAHAL_t ** old_phy )
```

Register/Replace a HAL with a.

Parameters

in	*iface_type*	- the type of physical interface to register
in	*hal*	pointer to the new ATCAHAL_t structure to register
out	*old*	pointer to the existing ATCAHAL_t structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.26 hal_iface_release()

```
ATCA_STATUS hal_iface_release (
            ATCAIfaceType iface_type,
            void * hal_data )
```

releases a physical interface, HAL knows how to interpret hal_data

Parameters

in	*iface_type*	- the type of physical interface to release
in	*hal_data*	- pointer to opaque hal data maintained by HAL implementation for this interface type

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.27 hal_is_command_word()

```
uint8_t hal_is_command_word (
            uint8_t word_address )
```

Utility function for hal_wake to check the reply.

Parameters

in	*word_address*	Command to check

Returns

true if the word_address is considered a command

8.8.5.28 hal_kit_attach_phy()

```
ATCA_STATUS hal_kit_attach_phy (
            ATCAIfaceCfg * cfg,
            atca_hal_kit_phy_t * phy )
```

Helper function that connects a physical layer context structure that will be used by the kit protocol bridge.

Returns

ATCA_STATUS

Parameters

cfg	[IN] Interface configuration structure
phy	[IN] Structure with physical layer interface functions and context

8.8.5.29 hal_kit_control()

```
ATCA_STATUS hal_kit_control (
            ATCAIface iface,
            uint8_t option )
```

Kit Protocol Control.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over
in	*option*	Control option to use

Returns

ATCA_STATUS

8.8.5.30 hal_kit_discover_buses()

```
ATCA_STATUS hal_kit_discover_buses (
            int busses[],
            int max_buses )
```

Request a list of busses from the kit host.

8.8.5.31 hal_kit_discover_devices()

```
ATCA_STATUS hal_kit_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

8.8.5.32 hal_kit_hid_control()

```
ATCA_STATUS hal_kit_hid_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.33 hal_kit_hid_init()

```
ATCA_STATUS hal_kit_hid_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

HAL implementation of Kit USB HID init.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_STATUS

8.8.5.34 hal_kit_hid_post_init()

```
ATCA_STATUS hal_kit_hid_post_init (
            ATCAIface iface )
```

HAL implementation of Kit HID post init.

Parameters

in	*iface*	instance

Returns

ATCA_STATUS

8.8.5.35 hal_kit_hid_receive()

```
ATCA_STATUS hal_kit_hid_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation of send over USB HID.

Parameters

in	*iface*	instance
in	*word_address*	determine device transaction type
in	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_STATUS

8.8.5.36 hal_kit_hid_release()

```
ATCA_STATUS hal_kit_hid_release (
            void * hal_data )
```

Close the physical port for HID.

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_STATUS

8.8.5.37 hal_kit_hid_send()

```
ATCA_STATUS hal_kit_hid_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send over USB HID.

Parameters

in	*iface*	instance
in	*word_address*	determine device transaction type
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_STATUS

8.8.5.38 hal_kit_init()

```
ATCA_STATUS hal_kit_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

HAL implementation of Kit USB HID init.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_STATUS

8.8.5.39 hal_kit_post_init()

```
ATCA_STATUS hal_kit_post_init (
            ATCAIface iface )
```

HAL implementation of Kit HID post init.

Parameters

in	*iface*	instance

Returns

ATCA_STATUS

8.8.5.40 hal_kit_receive()

```
ATCA_STATUS hal_kit_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation of send over USB HID.

Parameters

in	*iface*	instance
in	*word_address*	determine device transaction type
in	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_STATUS

8.8.5.41 hal_kit_release()

```
ATCA_STATUS hal_kit_release (
            void * hal_data )
```

Close the physical port for HID.

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_STATUS

8.8.5.42 hal_kit_send()

```
ATCA_STATUS hal_kit_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send over USB HID.

Parameters

in	*iface*	instance
in	*word_address*	determine device transaction type
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_STATUS

8.8.5.43 hal_lock_mutex()

```
ATCA_STATUS hal_lock_mutex (
            void * pMutex )
```

8.8.5.44 hal_malloc()

```
void * hal_malloc (
            size_t size )
```

8.8.5.45 hal_rtos_delay_ms()

```
void hal_rtos_delay_ms (
            uint32_t delay )
```

Timer API implemented at the HAL level.

This function delays for a number of milliseconds.

```
You can override this function if you like to do
something else in your system while delaying.
```

Parameters

in	*delay*	Number of milliseconds to delay

8.8.5.46 hal_spi_control()

```
ATCA_STATUS hal_spi_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.47 hal_spi_deselect()

```
ATCA_STATUS hal_spi_deselect (
            ATCAIface iface )
```

HAL implementation to deassert the device chip select.

Parameters

in	*iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.48 hal_spi_discover_buses()

```
ATCA_STATUS hal_spi_discover_buses (
            int spi_buses[ ],
            int max_buses )
```

discover spi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*spi_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

8.8.5.49 hal_spi_discover_devices()

```
ATCA_STATUS hal_spi_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any TA100 devices on a given logical bus number

Parameters

in	*bus_num*	logical bus number on which to look for TA100 devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_SUCCESS

8.8.5.50 hal_spi_init()

```
ATCA_STATUS hal_spi_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

initialize an SPI interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.51 hal_spi_post_init()

```
ATCA_STATUS hal_spi_post_init (
            ATCAIface iface )
```

HAL implementation of SPI post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

8.8.5.52 hal_spi_receive()

```
ATCA_STATUS hal_spi_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of SPI receive function for HARMONY SPI.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.53 hal_spi_release()

```
ATCA_STATUS hal_spi_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.54 hal_spi_select()

```
ATCA_STATUS hal_spi_select (
            ATCAIface iface )
```

HAL implementation to assert the device chip select.

Parameters

in	*iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.55 hal_spi_send()

```
ATCA_STATUS hal_spi_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of SPI send over Harmony.

Parameters

in	*iface*	instance
in	*word_address*	device transaction type
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.56 hal_swi_control()

```
ATCA_STATUS hal_swi_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.57 hal_swi_discover_buses()

```
ATCA_STATUS hal_swi_discover_buses (
            int swi_buses[ ],
            int max_buses )
```

discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge

Parameters

in	*swi_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

8.8.5.58 hal_swi_discover_devices()

```
ATCA_STATUS hal_swi_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

8.8.5.59 hal_swi_idle()

```
ATCA_STATUS hal_swi_idle (
            ATCAIface iface )
```

Send Idle flag via SWI.

Parameters

in	*iface*	interface of the logical device to idle

Returns

ATCA_SUCCES

8.8.5.60 hal_swi_init() [1/2]

```
ATCA_STATUS hal_swi_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

initialize an SWI interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.61 hal_swi_init() [2/2]

```
ATCA_STATUS hal_swi_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_↩ swi_init manages these things and ATCAIFace is abstracted from the physical details.

Initialize an SWI interface using given config.

Parameters

in	*hal*	opaque pointer to HAL data
in	*cfg*	interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.62 hal_swi_post_init()

```
ATCA_STATUS hal_swi_post_init (
            ATCAIface iface )
```

HAL implementation of SWI post init.

Parameters

in	*iface*	ATCAIface instance

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

8.8.5.63 hal_swi_receive()

```
ATCA_STATUS hal_swi_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

Receive byte(s) via SWI.

HAL implementation of SWI receive function over UART.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.64 hal_swi_release()

```
ATCA_STATUS hal_swi_release (
            void * hal_data )
```

Manages reference count on given bus and releases resource if no more reference(s) exist.

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.65 hal_swi_send()

```
ATCA_STATUS hal_swi_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

Send byte(s) via SWI.

HAL implementation of SWI send command over UART.

Parameters

in	*iface*	interface of the logical device to send data to
in	*word_address*	device transaction type
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance
in	*word_address*	device transaction type
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Send Command Flag

Skip the Word Address data as SWI doesn't use it

Send the remaining bytes

8.8.5.66 hal_swi_sleep()

```
ATCA_STATUS hal_swi_sleep (
            ATCAIface iface )
```

Send Sleep flag via SWI.

Parameters

in	*iface*	interface of the logical device to sleep

Returns

ATCA_SUCCESS

8.8.5.67 hal_swi_wake()

```
ATCA_STATUS hal_swi_wake (
            ATCAIface iface )
```

Send Wake flag via SWI.

Parameters

in	*iface*	interface of the logical device to wake up

Returns

ATCA_SUCCESS on success, otherwise an error code.

Generate Wake Token

Wait tWHI + tWLO

8.8.5.68 hal_unlock_mutex()

```
ATCA_STATUS hal_unlock_mutex (
            void * pMutex )
```

8.8.5.69 kit_control()

```
ATCA_STATUS kit_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

8.8.5.70 kit_id_from_devtype()

```
const char * kit_id_from_devtype (
            ATCADeviceType devtype )
```

Kit Protocol is key

8.8.5.71 kit_idle()

```
ATCA_STATUS kit_idle (
            ATCAIface iface )
```

8.8.5.72 kit_init()

```
ATCA_STATUS kit_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

8.8.5.73 kit_interface_from_kittype()

```
const char * kit_interface_from_kittype (
            ATCAKitType kittype )
```

Kit interface from device

8.8.5.74 kit_parse_rsp()

```
ATCA_STATUS kit_parse_rsp (
            const char * pkitbuf,
            int nkitbuf,
            uint8_t * kitstatus,
            uint8_t * rxdata,
            int * nrxdata )
```

8.8.5.75 kit_post_init()

```
ATCA_STATUS kit_post_init (
            ATCAIface iface )
```

8.8.5.76 kit_receive()

```
ATCA_STATUS kit_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

8.8.5.77 kit_release()

```
ATCA_STATUS kit_release (
            void * hal_data )
```

8.8.5.78 kit_send()

```
ATCA_STATUS kit_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

8.8.5.79 kit_sleep()

```
ATCA_STATUS kit_sleep (
            ATCAIface iface )
```

8.8.5.80 kit_wake()

```
ATCA_STATUS kit_wake (
            ATCAIface iface )
```

8.8.5.81 kit_wrap_cmd()

```
ATCA_STATUS kit_wrap_cmd (
            const uint8_t * txdata,
            int txlength,
            char * pkitbuf,
            int * nkitbuf,
            char target )
```

8.8.5.82 strnchr()

```
char* strnchr (
            const char * s,
            size_t count,
            int c )
```

8.8.5.83 swi_uart_deinit()

```
ATCA_STATUS swi_uart_deinit (
            ATCASWIMaster_t * instance )
```

Implementation of SWI UART deinit.

HAL implementation of SWI UART deinit.

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS

8.8.5.84 swi_uart_discover_buses()

```
void swi_uart_discover_buses (
            int swi_uart_buses[],
            int max_buses )
```

discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*swi_uart_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

8.8.5.85 swi_uart_init()

```
ATCA_STATUS swi_uart_init (
            ATCASWIMaster_t * instance )
```

Implementation of SWI UART init.

HAL implementation of SWI UART init.

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the START SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the START UART drivers are a dependency *

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.86 swi_uart_mode()

```
void swi_uart_mode (
            ATCASWIMaster_t * instance,
            uint8_t mode )
```

implementation of SWI UART change mode.

HAL implementation of SWI UART change mode.

Parameters

in	*instance*	instance
in	*mode*	(TRANSMIT_MODE or RECEIVE_MODE)

8.8.5.87 swi_uart_receive_byte()

```
ATCA_STATUS swi_uart_receive_byte (
            ATCASWIMaster_t * instance,
            uint8_t * data )
```

HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Parameters

in	*instance*	instance
out	*data*	pointer to space to receive the data

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.88 swi_uart_send_byte()

```
ATCA_STATUS swi_uart_send_byte (
            ATCASWIMaster_t * instance,
            uint8_t data )
```

HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.

Parameters

in	*instance*	instance
in	*data*	number of byte to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.8.5.89 swi_uart_setbaud()

```
void swi_uart_setbaud (
            ATCASWIMaster_t * instance,
            uint32_t baudrate )
```

implementation of SWI UART change baudrate.

HAL implementation of SWI UART change baudrate.

Parameters

in	*instance*	instance
in	*baudrate*	(typically 230400 , 160000 or 115200)
in	*instance*	instance
in	*baudrate*	(typically 230400 or 115200)

8.8.6 Variable Documentation

8.8.6.1 pin_conf

```
struct port_config pin_conf
```

8.9 Host side crypto methods (atcah_)

Use these functions if your system does not use an ATCADevice as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions. They can be called directly from an application, or integrated into an API.

Data Structures

- struct atca_temp_key

  Structure to hold TempKey fields.
- struct atca_include_data_in_out

  Input / output parameters for function atca_include_data().
- struct atca_nonce_in_out

  Input/output parameters for function atca_nonce().
- struct atca_io_decrypt_in_out
- struct atca_verify_mac
- struct atca_secureboot_enc_in_out
- struct atca_secureboot_mac_in_out
- struct atca_mac_in_out

  Input/output parameters for function atca_mac().
- struct atca_hmac_in_out

  Input/output parameters for function atca_hmac().
- struct atca_gen_dig_in_out

  Input/output parameters for function atcah_gen_dig().
- struct atca_write_mac_in_out

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- struct atca_derive_key_in_out

  Input/output parameters for function atcah_derive_key().
- struct atca_derive_key_mac_in_out

  Input/output parameters for function atcah_derive_key_mac().
- struct atca_decrypt_in_out

  Input/output parameters for function atca_decrypt().
- struct atca_check_mac_in_out

  Input/output parameters for function atcah_check_mac().
- struct atca_verify_in_out

  Input/output parameters for function atcah_verify().
- struct atca_gen_key_in_out

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- struct atca_sign_internal_in_out

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.
- struct atca_session_key_in_out

  Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

Typedefs

- typedef struct atca_temp_key atca_temp_key_t

    Structure to hold TempKey fields.
- typedef struct atca_nonce_in_out atca_nonce_in_out_t
- typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
- typedef struct atca_verify_mac atca_verify_mac_in_out_t
- typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
- typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
- typedef struct atca_mac_in_out atca_mac_in_out_t
- typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t

    Input/output parameters for function atcah_gen_dig().
- typedef struct atca_write_mac_in_out atca_write_mac_in_out_t

    Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- typedef struct atca_check_mac_in_out atca_check_mac_in_out_t

    Input/output parameters for function atcah_check_mac().
- typedef struct atca_verify_in_out atca_verify_in_out_t
- typedef struct atca_gen_key_in_out atca_gen_key_in_out_t

    Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t

    Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.
- typedef struct atca_session_key_in_out atca_session_key_in_out_t

    Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

Functions

- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out ∗param)

    This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out ∗param)

    This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out ∗param)

    This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out ∗param)

    This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out ∗param)

    This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out ∗param)

    This function generates mac with session key with a plain text.
- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out ∗param)

    This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out ∗param)

    This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out ∗param)

    This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out ∗param)

    This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out ∗param)

    This function decrypts 32-byte encrypted data received with the Read command.

- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t ∗message, uint8_t ∗digest)

    This function creates a SHA256 digest on a little-endian system.
- uint8_t ∗ atcah_include_data (struct atca_include_data_in_out ∗param)

    This function copies otp and sn data into a command buffer.
- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out ∗param)

    Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param, const uint8_t ∗config)

    Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param)

    Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t ∗param)

    Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t ∗param)

    Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t ∗param)

    Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter, uint8_t ∗counter_match)

    Builds the counter match value that needs to be stored in a slot.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out ∗param)

    Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608 are the only ones that support this operation.
- ATCA_STATUS atcah_ecc204_write_auth_mac (struct atca_write_mac_in_out ∗param)

    This function calculates the input MAC for the ECC204 Write command.
- ATCA_STATUS atcah_gen_session_key (atca_session_key_in_out_t ∗param)

    This function calculates the session key for the ECC204.

Variables

- uint8_t ∗ p_temp

    [out] pointer to output buffer
- const uint8_t ∗ otp

    [in] pointer to one-time-programming data
- const uint8_t ∗ sn

    [in] pointer to serial number data
- uint8_t mode

    [in] Mode parameter used in Nonce command (Param1).
- uint16_t zero

    [in] Zero parameter used in Nonce command (Param2).
- const uint8_t ∗ num_in

    [in] Pointer to 20-byte NumIn data used in Nonce command.
- const uint8_t ∗ rand_out

    [in] Pointer to 32-byte RandOut data from Nonce command.
- struct atca_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.
- uint8_t mode

    [in] Mode parameter used in MAC command (Param1).
- uint16_t key_id

[in] KeyID parameter used in MAC command (Param2).

- const uint8_t * challenge

  [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.
- const uint8_t * key

  [in] Pointer to 32-byte key used to generate MAC digest.
- const uint8_t * otp

  [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- const uint8_t * sn

  [in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.
- uint8_t * response

  [out] Pointer to 32-byte SHA-256 digest (MAC).
- struct atca_temp_key * temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t mode

  [in] Mode parameter used in HMAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in HMAC command (Param2).
- const uint8_t * key

  [in] Pointer to 32-byte key used to generate HMAC digest.
- const uint8_t * otp

  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.
- const uint8_t * sn

  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.
- uint8_t * response

  [out] Pointer to 32-byte SHA-256 HMAC digest.
- struct atca_temp_key * temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t * crypto_data

  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct atca_temp_key * temp_key

  [in,out] Pointer to TempKey structure.
- uint16_t curve_type

  [in] Curve type used in Verify command (Param2).
- const uint8_t * signature

  [in] Pointer to ECDSA signature to be verified
- const uint8_t * public_key

  [in] Pointer to the public key to be used for verification
- struct atca_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

Definitions for ATECC Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define ATCA_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.
- #define ATCA_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}

- #define ATCA_MSG_SIZE_HMAC (88)
- #define ATCA_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.
- #define ATCA_MSG_SIZE_ENCRYPT_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_SESSION_KEY (96)

  TransportKey{32} || 0x15{1} || 0x00{1} || KeyId{2} || SN8{1} || SN0_1{2} || 0{25} || Nonce{32}.
- #define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.
- #define ATCA_COMMAND_HEADER_SIZE (4)
- #define ATCA_GENDIG_ZEROS_SIZE (25)
- #define ATCA_WRITE_MAC_ZEROS_SIZE (25)
- #define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
- #define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
- #define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
- #define ATCA_HMAC_BLOCK_SIZE (64)
- #define ENCRYPTION_KEY_SIZE (64)

Default Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define ATCA_SN_0_DEF (0x01)
- #define ATCA_SN_1_DEF (0x23)
- #define ATCA_SN_8_DEF (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)

  mode mask for MAC command when using TempKey

8.9.1 Detailed Description

Use these functions if your system does not use an ATCADevice as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions. They can be called directly from an application, or integrated into an API.

Modern compilers can garbage-collect unused functions. If your compiler does not support this feature, you can just discard this module from your project if you do use an ATECC as a host. Or, if you don't, delete the functions you do not use.

8.9.2 Macro Definition Documentation

8.9.2.1 ATCA_COMMAND_HEADER_SIZE

```
#define ATCA_COMMAND_HEADER_SIZE ( 4)
```

8.9.2.2 ATCA_DERIVE_KEY_ZEROS_SIZE

```
#define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
```

8.9.2.3 ATCA_GENDIG_ZEROS_SIZE

```
#define ATCA_GENDIG_ZEROS_SIZE (25)
```

8.9.2.4 ATCA_HMAC_BLOCK_SIZE

```
#define ATCA_HMAC_BLOCK_SIZE (64)
```

8.9.2.5 ATCA_MSG_SIZE_DERIVE_KEY

```
#define ATCA_MSG_SIZE_DERIVE_KEY (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

8.9.2.6 ATCA_MSG_SIZE_DERIVE_KEY_MAC

```
#define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.

8.9.2.7 ATCA_MSG_SIZE_ENCRYPT_MAC

```
#define ATCA_MSG_SIZE_ENCRYPT_MAC (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

8.9.2.8 ATCA_MSG_SIZE_GEN_DIG

```
#define ATCA_MSG_SIZE_GEN_DIG (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

8.9.2.9 ATCA_MSG_SIZE_HMAC

```
#define ATCA_MSG_SIZE_HMAC (88)
```

8.9.2.10 ATCA_MSG_SIZE_MAC

```
#define ATCA_MSG_SIZE_MAC (88)
```

(Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}

8.9.2.11 ATCA_MSG_SIZE_NONCE

```
#define ATCA_MSG_SIZE_NONCE (55)
```

RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.

8.9.2.12 ATCA_MSG_SIZE_PRIVWRITE_MAC

```
#define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.

8.9.2.13 ATCA_MSG_SIZE_SESSION_KEY

```
#define ATCA_MSG_SIZE_SESSION_KEY (96)
```

TransportKey{32} || 0x15{1} || 0x00{1} || KeyId{2} || SN8{1} || SN0_1{2} || 0{25} || Nonce{32}.

8.9.2.14 ATCA_PRIVWRITE_MAC_ZEROS_SIZE

```
#define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
```

8.9.2.15 ATCA_PRIVWRITE_PLAIN_TEXT_SIZE

```
#define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
```

8.9.2.16 ATCA_SN_0_DEF

```
#define ATCA_SN_0_DEF (0x01)
```

8.9.2.17 ATCA_SN_1_DEF

```
#define ATCA_SN_1_DEF (0x23)
```

8.9.2.18 ATCA_SN_8_DEF

```
#define ATCA_SN_8_DEF (0xEE)
```

8.9.2.19 ATCA_WRITE_MAC_ZEROS_SIZE

```
#define ATCA_WRITE_MAC_ZEROS_SIZE (25)
```

8.9.2.20 ENCRYPTION_KEY_SIZE

```
#define ENCRYPTION_KEY_SIZE (64)
```

8.9.2.21 MAC_MODE_USE_TEMPKEY_MASK

```
#define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)
```

mode mask for MAC command when using TempKey

8.9.3 Typedef Documentation

8.9.3.1 atca_check_mac_in_out_t

```
typedef struct atca_check_mac_in_out atca_check_mac_in_out_t
```

Input/output parameters for function atcah_check_mac().

8.9.3.2 atca_gen_dig_in_out_t

```
typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t
```

Input/output parameters for function atcah_gen_dig().

8.9.3.3 atca_gen_key_in_out_t

```
typedef struct atca_gen_key_in_out atca_gen_key_in_out_t
```

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

8.9.3.4 atca_io_decrypt_in_out_t

```
typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
```

8.9.3.5 atca_mac_in_out_t

```
typedef struct atca_mac_in_out atca_mac_in_out_t
```

8.9.3.6 atca_nonce_in_out_t

```
typedef struct atca_nonce_in_out atca_nonce_in_out_t
```

8.9.3.7 atca_secureboot_enc_in_out_t

```
typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
```

8.9.3.8 atca_secureboot_mac_in_out_t

```
typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
```

8.9.3.9 atca_session_key_in_out_t

```
typedef struct atca_session_key_in_out atca_session_key_in_out_t
```

Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

8.9.3.10 atca_sign_internal_in_out_t

```
typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t
```

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

8.9.3.11 atca_temp_key_t

```
typedef struct atca_temp_key atca_temp_key_t
```

Structure to hold TempKey fields.

8.9.3.12 atca_verify_in_out_t

```
typedef struct atca_verify_in_out atca_verify_in_out_t
```

8.9.3.13 atca_verify_mac_in_out_t

```
typedef struct atca_verify_mac atca_verify_mac_in_out_t
```

8.9.3.14 atca_write_mac_in_out_t

```
typedef struct atca_write_mac_in_out atca_write_mac_in_out_t
```

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

8.9.4 Function Documentation

8.9.4.1 atcah_check_mac()

```
ATCA_STATUS atcah_check_mac (
            struct atca_check_mac_in_out * param )
```

This function performs the checkmac operation to generate client response on the host side .

Parameters

in,out	*param*	Input and output parameters

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.2 atcah_config_to_sign_internal()

```
ATCA_STATUS atcah_config_to_sign_internal (
            ATCADeviceType device_type,
            struct atca_sign_internal_in_out * param,
            const uint8_t * config )
```

Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.

The atca_sign_internal_in_out structure has a number of fields (slot_config, key_config, is_slot_locked) that can be determined automatically from the current state of TempKey and the full config zone.

Parameters

in,out	*param*	Sign(Internal) parameters to be filled out. Only slot_config, key_config, and is_slot_locked will be set.
in	*device_type*	The type of the device.
in	*config*	Full 128 byte config zone for the device.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.3 atcah_decrypt()

```
ATCA_STATUS atcah_decrypt (
            struct atca_decrypt_in_out * param )
```

This function decrypts 32-byte encrypted data received with the Read command.

To use this function, first the nonce must be valid and synchronized between device and application. The application sends a GenDig command to the Device, using a key specified by SlotConfig.ReadKey. The device updates its TempKey. The application then updates its own TempKey using the GenDig calculation function, using the same key. The application sends a Read command to the device for a user zone configured with EncryptRead. The device encrypts 32-byte zone content, and outputs it to the host. The application passes these encrypted data to this decryption function. The function decrypts the data and returns them. TempKey must be updated by Gen↩Dig using a ParentKey as specified by SlotConfig.ReadKey before executing this function. The decryption function does not check whether the TempKey has been generated by a correct ParentKey for the corresponding zone. Therefore to get a correct result, the application has to make sure that prior GenDig calculation was done using correct ParentKey.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.4 atcah_derive_key()

```
ATCA_STATUS atcah_derive_key (
            struct atca_derive_key_in_out * param )
```

This function derives a key with a key and TempKey.

Used in conjunction with DeriveKey command, the key derived by this function will match the key in the device. Two kinds of operation are supported:

- Roll Key operation: target_key and parent_key parameters should be set to point to the same location (TargetKey).
- Create Key operation: target_key should be set to point to TargetKey, parent_key should be set to point to ParentKey.

After executing this function, the initial value of target_key will be overwritten with the derived key. The TempKey should be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.5 atcah_derive_key_mac()

```
ATCA_STATUS atcah_derive_key_mac (
            struct atca_derive_key_mac_in_out * param )
```

This function calculates the input MAC for a DeriveKey command.

The DeriveKey command will need an input MAC if SlotConfig[TargetKey].Bit15 is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.6 atcah_ecc204_write_auth_mac()

```
ATCA_STATUS atcah_ecc204_write_auth_mac (
            struct atca_write_mac_in_out * param )
```

This function calculates the input MAC for the ECC204 Write command.

The Write command will need an input MAC if SlotConfig3.bit0 is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.7 atcah_encode_counter_match()

```
ATCA_STATUS atcah_encode_counter_match (
            uint32_t counter_value,
            uint8_t * counter_match_value )
```

Builds the counter match value that needs to be stored in a slot.

Parameters

in	*counter_value*	Counter value to be used for the counter match. This must be a multiple of 32.
out	*counter_match_value*	Data to be stored in the beginning of a counter match slot will be returned here (8 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.8 atcah_gen_dig()

```
ATCA_STATUS atcah_gen_dig (
            struct atca_gen_dig_in_out * param )
```

This function combines the current TempKey with a stored value.

The stored value can be a data slot, OTP page, configuration zone, or hardware transport key. The TempKey generated by this function will match with the TempKey in the device generated when executing a GenDig command. The TempKey should be valid (temp_key.valid = 1) before executing this function. To use this function, an application first sends a GenDig command with a chosen stored value to the device. This stored value must be known by the application and is passed to this GenDig calculation function. The function calculates a new TempKey and returns it.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.9 atcah_gen_key_msg()

```
ATCA_STATUS atcah_gen_key_msg (
            struct atca_gen_key_in_out * param )
```

Calculate the PubKey digest created by GenKey and saved to TempKey.

Parameters

in,out	*param*	GenKey parameters required to calculate the PubKey digest. Digest is return in the temp_key parameter.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.10 atcah_gen_mac()

```
ATCA_STATUS atcah_gen_mac (
            struct atca_gen_dig_in_out * param )
```

This function generates mac with session key with a plain text.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.11 atcah_gen_session_key()

```
ATCA_STATUS atcah_gen_session_key (
            struct atca_session_key_in_out * param )
```

This function calculates the session key for the ECC204.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.12 atcah_hmac()

```
ATCA_STATUS atcah_hmac (
            struct atca_hmac_in_out * param )
```

This function generates an HMAC / SHA-256 hash of a key and other information.

The resulting hash will match with the one generated in the device by an HMAC command. The TempKey has to be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.13 atcah_include_data()

```
uint8_t* atcah_include_data (
            struct atca_include_data_in_out * param )
```

This function copies otp and sn data into a command buffer.

Parameters

in,out	*param*	pointer to parameter structure

Returns

pointer to command buffer byte that was copied last

8.9.4.14 atcah_io_decrypt()

```
ATCA_STATUS atcah_io_decrypt (
            struct atca_io_decrypt_in_out * param )
```

Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608 are the only ones that support this operation.

Parameters

in,out	*param*	Parameters required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.15 atcah_mac()

```
ATCA_STATUS atcah_mac (
            struct atca_mac_in_out * param )
```

This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.

The resulting digest will match with the one generated by the device when executing a MAC command. The Temp↩ Key (if used) should be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.16 atcah_nonce()

```
ATCA_STATUS atcah_nonce (
            struct atca_nonce_in_out * param )
```

This function calculates host side nonce with the parameters passed.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.17 atcah_privwrite_auth_mac()

```
ATCA_STATUS atcah_privwrite_auth_mac (
            struct atca_write_mac_in_out * param )
```

This function calculates the input MAC for the PrivWrite command.

The PrivWrite command will need an input MAC if SlotConfig.WriteConfig.Encrypt is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.18 atcah_secureboot_enc()

```
ATCA_STATUS atcah_secureboot_enc (
            atca_secureboot_enc_in_out_t * param )
```

Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.19 atcah_secureboot_mac()

```
ATCA_STATUS atcah_secureboot_mac (
            atca_secureboot_mac_in_out_t * param )
```

Calculates the expected MAC returned from the SecureBoot command when verification is a success.

The result of this function (param->mac) should be compared with the actual MAC returned to validate the response.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.20 atcah_sha256()

```
ATCA_STATUS atcah_sha256 (
            int32_t len,
            const uint8_t * message,
            uint8_t * digest )
```

This function creates a SHA256 digest on a little-endian system.

Parameters

in	*len*	byte length of message
in	*message*	pointer to message
out	*digest*	SHA256 of message

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.21 atcah_sign_internal_msg()

```
ATCA_STATUS atcah_sign_internal_msg (
            ATCADeviceType device_type,
            struct atca_sign_internal_in_out * param )
```

Builds the full message that would be signed by the Sign(Internal) command.

Additionally, the function will optionally output the OtherData data required by the Verify(In/Validate) command as well as the SHA256 digest of the full message.

Parameters

out	*device_type*	Device type to perform the calculation for.
out	*param*	Input data and output buffers required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.22 atcah_verify_mac()

```
ATCA_STATUS atcah_verify_mac (
            atca_verify_mac_in_out_t * param )
```

Calculate the expected MAC on the host side for the Verify command.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.4.23 atcah_write_auth_mac()

```
ATCA_STATUS atcah_write_auth_mac (
             struct atca_write_mac_in_out * param )
```

This function calculates the input MAC for the Write command.

The Write command will need an input MAC if SlotConfig.WriteConfig.Encrypt is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.9.5 Variable Documentation

8.9.5.1 challenge

```
challenge
```

[in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.

8.9.5.2 crypto_data

```
crypto_data
```

[in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.

8.9.5.3 curve_type

```
curve_type
```

[in] Curve type used in Verify command (Param2).

8.9.5.4 key [1/2]

key

[in] Pointer to 32-byte key used to generate MAC digest.

8.9.5.5 key [2/2]

key

[in] Pointer to 32-byte key used to generate HMAC digest.

8.9.5.6 key_id [1/2]

key_id

[in] KeyID parameter used in MAC command (Param2).

8.9.5.7 key_id [2/2]

key_id

[in] KeyID parameter used in HMAC command (Param2).

8.9.5.8 mode [1/3]

mode

[in] Mode parameter used in Nonce command (Param1).

8.9.5.9 mode [2/3]

mode

[in] Mode parameter used in MAC command (Param1).

8.9.5.10 mode [3/3]

`mode`

[in] Mode parameter used in HMAC command (Param1).

8.9.5.11 num_in

`num_in`

[in] Pointer to 20-byte NumIn data used in Nonce command.

8.9.5.12 otp [1/3]

`otp`

[in] pointer to one-time-programming data

8.9.5.13 otp [2/3]

`otp`

[in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.

8.9.5.14 otp [3/3]

`otp`

[in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.

8.9.5.15 p_temp

`p_temp`

[out] pointer to output buffer

8.9.5.16 public_key

`public_key`

[in] Pointer to the public key to be used for verification

8.9.5.17 rand_out

`rand_out`

[in] Pointer to 32-byte RandOut data from Nonce command.

8.9.5.18 response [1/2]

`response`

[out] Pointer to 32-byte SHA-256 digest (MAC).

8.9.5.19 response [2/2]

`response`

[out] Pointer to 32-byte SHA-256 HMAC digest.

8.9.5.20 signature

`signature`

[in] Pointer to ECDSA signature to be verified

8.9.5.21 sn [1/3]

`sn`

[in] pointer to serial number data

8.9.5.22 sn [2/3]

sn

[in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.

8.9.5.23 sn [3/3]

sn

[in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.

8.9.5.24 temp_key [1/5]

temp_key

[in,out] Pointer to TempKey structure.

8.9.5.25 temp_key [2/5]

temp_key

[in,out] Pointer to TempKey structure.

8.9.5.26 temp_key [3/5]

temp_key

[in,out] Pointer to TempKey structure.

8.9.5.27 temp_key [4/5]

temp_key

[in,out] Pointer to TempKey structure.

8.9.5.28 temp_key [5/5]

temp_key

[in,out] Pointer to TempKey structure.

8.9.5.29 zero

zero

[in] Zero parameter used in Nonce command (Param2).

8.10 JSON Web Token (JWT) methods (atca_jwt_)

Methods for signing and verifying JSON Web Token (JWT) tokens.

Data Structures

- struct atca_jwt_t
  Structure to hold metadata information about the jwt being built.

Functions

- ATCA_STATUS atca_jwt_init (atca_jwt_t ∗jwt, char ∗buf, uint16_t buflen)
  Initialize a JWT structure.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t ∗jwt, const char ∗claim, const char ∗value)
  Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t ∗jwt, const char ∗claim, int32_t value)
  Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t ∗jwt, uint16_t key_id)
  Close the claims of a token, encode them, then sign the result.
- void atca_jwt_check_payload_start (atca_jwt_t ∗jwt)
  Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_verify (const char ∗buf, uint16_t buflen, const uint8_t ∗pubkey)
  Verifies the signature of a jwt using the provided public key.

8.10.1 Detailed Description

Methods for signing and verifying JSON Web Token (JWT) tokens.

8.10.2 Function Documentation

8.10.2.1 atca_jwt_add_claim_numeric()

```
ATCA_STATUS atca_jwt_add_claim_numeric (
            atca_jwt_t * jwt,
            const char * claim,
            int32_t value )
```

Add a numeric claim to a token.

Note

This function does not escape strings so the user has to ensure the claim is valid first

Parameters

in	*jwt*	JWT Context to use
in	*claim*	Name of the claim to be inserted
in	*value*	integer value to be inserted

8.10.2.2 atca_jwt_add_claim_string()

```
ATCA_STATUS atca_jwt_add_claim_string (
            atca_jwt_t * jwt,
            const char * claim,
            const char * value )
```

Add a string claim to a token.

Note

This function does not escape strings so the user has to ensure they are valid for use in a JSON string first

Parameters

in	*jwt*	JWT Context to use
in	*claim*	Name of the claim to be inserted
in	*value*	Null terminated string to be insterted

8.10.2.3 atca_jwt_check_payload_start()

```
void atca_jwt_check_payload_start (
            atca_jwt_t * jwt )
```

Check the provided context to see what character needs to be added in order to append a claim.

Parameters

in	*jwt*	JWT Context to use

8.10.2.4 atca_jwt_finalize()

```
ATCA_STATUS atca_jwt_finalize (
            atca_jwt_t * jwt,
            uint16_t key_id )
```

Close the claims of a token, encode them, then sign the result.

Parameters

in	*jwt*	JWT Context to use
in	*key←_id*	Key Id (Slot number) used to sign

8.10.2.5 atca_jwt_init()

```
ATCA_STATUS atca_jwt_init (
            atca_jwt_t * jwt,
            char * buf,
            uint16_t buflen )
```

Initialize a JWT structure.

Parameters

in	*jwt*	JWT Context to initialize
in,out	*buf*	Pointer to a buffer to store the token
in	*buflen*	Length of the buffer

8.10.2.6 atca_jwt_verify()

```
ATCA_STATUS atca_jwt_verify (
            const char * buf,
            uint16_t buflen,
            const uint8_t * pubkey )
```

Verifies the signature of a jwt using the provided public key.

Parameters

in	*buf*	Buffer holding an encoded jwt
in	*buflen*	Length of the buffer/jwt
in	*pubkey*	Public key (raw byte format)

8.11 mbedTLS Wrapper methods (atca_mbedtls_)

These methods are for interfacing cryptoauthlib to mbedtls.

8.11.0.1 mbedtls directory - Purpose

This directory contains the interfacing and wrapper functions to integrate mbedtls as the software crypto library as well as provide eliptic curve cryptography (ECC) hardware acceleration.

Data Structures

- struct atca_mbedtls_eckey_s

Typedefs

- typedef struct atca_mbedtls_eckey_s atca_mbedtls_eckey_t

Functions

- int atca_mbedtls_ecdsa_sign (const mbedtls_mpi ∗d, mbedtls_mpi ∗r, mbedtls_mpi ∗s, const unsigned char ∗buf, size_t buf_len)
- int atca_mbedtls_pk_init_ext (ATCADevice device, struct mbedtls_pk_context ∗pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_pk_init (struct mbedtls_pk_context ∗pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (struct mbedtls_x509_crt ∗cert, const struct atcacert_def_s ∗cert_def)
- int atca_mbedtls_ecdh_slot_cb (void)

  ECDH Callback to obtain the "slot" used in ECDH operations from the application.
- int atca_mbedtls_ecdh_ioprot_cb (uint8_t secret[32])

  ECDH Callback to obtain the IO Protection secret from the application.

8.11.1 Detailed Description

These methods are for interfacing cryptoauthlib to mbedtls.

8.11.2 Typedef Documentation

8.11.2.1 atca_mbedtls_eckey_t

```
typedef struct atca_mbedtls_eckey_s atca_mbedtls_eckey_t
```

Structure to hold metadata - is written into the mbedtls pk structure as the private key bignum value 'd' which otherwise would be unused. Bignums can be any arbitrary length of bytes

8.11.3 Function Documentation

8.11.3.1 atca_mbedtls_cert_add()

```
int atca_mbedtls_cert_add (
            struct mbedtls_x509_crt * cert,
            const struct atcacert_def_s * cert_def )
```

8.11.3.2 atca_mbedtls_ecdh_ioprot_cb()

```
int atca_mbedtls_ecdh_ioprot_cb (
            uint8_t secret[32] )
```

ECDH Callback to obtain the IO Protection secret from the application.

Parameters

out	*secret*	32 byte array used to store the secret

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.11.3.3 atca_mbedtls_ecdh_slot_cb()

```
int atca_mbedtls_ecdh_slot_cb (
            void )
```

ECDH Callback to obtain the "slot" used in ECDH operations from the application.

Returns

Slot Number

8.11.3.4 atca_mbedtls_ecdsa_sign()

```
int atca_mbedtls_ecdsa_sign (
            const mbedtls_mpi * d,
            mbedtls_mpi * r,
            mbedtls_mpi * s,
            const unsigned char * buf,
            size_t buf_len )
```

8.11.3.5 atca_mbedtls_pk_init()

```
int atca_mbedtls_pk_init (
            mbedtls_pk_context * pkey,
            const uint16_t slotid )
```

Initializes an mbedtls pk context for use with EC operations.

Parameters

in,out	*pkey*	ptr to space to receive version string
in	*slotid*	Associated with this key

Returns

0 on success, otherwise an error code.

8.11.3.6 atca_mbedtls_pk_init_ext()

```
int atca_mbedtls_pk_init_ext (
            ATCADevice device,
            mbedtls_pk_context * pkey,
            const uint16_t slotid )
```

Initializes an mbedtls pk context for use with EC operations.

Parameters

in,out	*pkey*	ptr to space to receive version string
in	*slotid*	Associated with this key

Returns

0 on success, otherwise an error code.

8.12 Attributes (pkcs11_attrib_)

Data Structures

- struct _pcks11_mech_table_e

Macros

- #define PCKS11_MECH_ECC508_EC_CAPABILITY (CKF_EC_F_P | CKF_EC_NAMEDCURVE | CKF_EC_UNCOMPRESS)
- #define TABLE_SIZE(x) sizeof(x) / sizeof(x[0])

Typedefs

- typedef struct _pcks11_mech_table_e pcks11_mech_table_e
- typedef struct _pcks11_mech_table_e ∗ pcks11_mech_table_ptr

Functions

- CK_RV pkcs11_attrib_fill (CK_ATTRIBUTE_PTR pAttribute, const CK_VOID_PTR pData, const CK_ULONG ulSize)

  Perform the nessasary checks and copy data into an attribute structure.
- CK_RV pkcs11_attrib_value (CK_ATTRIBUTE_PTR pAttribute, const CK_ULONG ulValue, const CK_ULONG ulSize)

  Helper function to write a numerical value to an attribute buffer.
- CK_RV pkcs11_attrib_false (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_true (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_empty (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_encoded (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_subject (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_subject_key_id (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_authority_key_id (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_trusted_flag (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_x509_write (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- void pkcs11_config_init_private (pkcs11_object_ptr pObject, char ∗label, size_t len)
- void pkcs11_config_init_public (pkcs11_object_ptr pObject, char ∗label, size_t len)
- void pkcs11_config_init_secret (pkcs11_object_ptr pObject, char ∗label, size_t len, uint8_t keylen)
- void pkcs11_config_init_cert (pkcs11_object_ptr pObject, char ∗label, size_t len)
- CK_RV pkcs11_config_cert (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr p←Object, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_key (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr p←Object, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_remove_object (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr pObject)
- CK_RV pkcs11_config_load_objects (pkcs11_slot_ctx_ptr slot_ctx)
- CK_RV pkcs11_config_load (pkcs11_slot_ctx_ptr slot_ctx)
- CK_RV pkcs11_encrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_encrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulData←Len, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

- CK_RV pkcs11_encrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
- CK_RV pkcs11_encrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

  Finishes a multiple-part encryption operation.
- CK_RV pkcs11_decrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_decrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)
- CK_RV pkcs11_decrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)
- CK_RV pkcs11_decrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

  Finishes a multiple-part decryption operation.
- CK_RV pkcs11_find_init (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_find_continue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount)
- CK_RV pkcs11_find_finish (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_find_get_attribute (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_get_lib_info (CK_INFO_PTR pInfo)

  Obtains general information about Cryptoki.
- pkcs11_lib_ctx_ptr pkcs11_get_context (void)

  Retrieve the current library context.
- CK_RV pkcs11_lock_context (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_unlock_context (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_init_check (pkcs11_lib_ctx_ptr ∗ppContext, CK_BBOOL lock)

  Check if the library is initialized properly.
- CK_RV pkcs11_init (CK_C_INITIALIZE_ARGS_PTR pInitArgs)

  Initializes the PKCS11 API Library for Cryptoauthlib.
- CK_RV pkcs11_deinit (CK_VOID_PTR pReserved)
- CK_RV pkcs11_key_write (CK_VOID_PTR pSession, CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR p←Attribute)
- CK_RV pkcs11_key_generate (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)
- CK_RV pkcs11_key_generate_pair (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p←Mechanism, CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR pPrivateKeyTemplate, CK_ULONG ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)
- CK_RV pkcs11_key_derive (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hBaseKey, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)
- CK_RV C_Initialize (CK_VOID_PTR pInitArgs)

  Initializes Cryptoki library NOTES: If pInitArgs is a non-NULL_PTR is must dereference to a CK_C_INITIALIZE_ARGS structure.
- CK_RV C_Finalize (CK_VOID_PTR pReserved)

  Clean up miscellaneous Cryptoki-associated resources.
- CK_RV C_GetInfo (CK_INFO_PTR pInfo)

  Obtains general information about Cryptoki.
- CK_RV C_GetFunctionList (CK_FUNCTION_LIST_PTR_PTR ppFunctionList)

  Obtains entry points of Cryptoki library functions.
- CK_RV C_GetSlotList (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pul←Count)

Obtains a list of slots in the system.

- CK_RV C_GetSlotInfo (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo)

  Obtains information about a particular slot.
- CK_RV C_GetTokenInfo (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo)

  Obtains information about a particular token.
- CK_RV C_GetMechanismList (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount)

  Obtains a list of mechanisms supported by a token (in a slot)
- CK_RV C_GetMechanismInfo (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo)

  Obtains information about a particular mechanism of a token (in a slot)
- CK_RV C_InitToken (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen, CK_UTF8CHAR_PTR pLabel)

  Initializes a token (in a slot)
- CK_RV C_InitPIN (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)

  Initializes the normal user's PIN.
- CK_RV C_SetPIN (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pOldPin, CK_ULONG ulOld↩ Len, CK_UTF8CHAR_PTR pNewPin, CK_ULONG ulNewLen)

  Modifies the PIN of the current user.
- CK_RV C_OpenSession (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_NOTIFY notify, CK_SESSION_HANDLE_PTR phSession)

  Opens a connection between an application and a particular token or sets up an application callback for token insertion.
- CK_RV C_CloseSession (CK_SESSION_HANDLE hSession)

  Close the given session.
- CK_RV C_CloseAllSessions (CK_SLOT_ID slotID)

  Close all open sessions.
- CK_RV C_GetSessionInfo (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo)

  Retrieve information about the specified session.
- CK_RV C_GetOperationState (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pOperationState, CK_ULONG_PTR pulOperationStateLen)

  Obtains the cryptographic operations state of a session.
- CK_RV C_SetOperationState (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pOperationState, CK_ULONG ulOperationStateLen, CK_OBJECT_HANDLE hEncryptionKey, CK_OBJECT_HANDLE h↩ AuthenticationKey)

  Sets the cryptographic operations state of a session.
- CK_RV C_Login (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)

  Login on the token in the specified session.
- CK_RV C_Logout (CK_SESSION_HANDLE hSession)

  Log out of the token in the specified session.
- CK_RV C_CreateObject (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phObject)

  Create a new object on the token in the specified session using the given attribute template.
- CK_RV C_CopyObject (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phNewObject)

  Create a copy of the object with the specified handle.
- CK_RV C_DestroyObject (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Destroy the specified object.
- CK_RV C_GetObjectSize (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ULONG_PTR pulSize)

  Obtains the size of an object in bytes.

- CK_RV C_GetAttributeValue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Obtains an attribute value of an object.
- CK_RV C_SetAttributeValue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Change or set the value of the specified attributes on the specified object.
- CK_RV C_FindObjectsInit (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Initializes an object search in the specified session using the specified attribute template as search parameters.
- CK_RV C_FindObjects (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount)

  Continue the search for objects in the specified session.
- CK_RV C_FindObjectsFinal (CK_SESSION_HANDLE hSession)

  Finishes an object search operation (and cleans up)
- CK_RV C_EncryptInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)

  Initializes an encryption operation using the specified mechanism and session.
- CK_RV C_Encrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

  Perform a single operation encryption operation in the specified session.
- CK_RV C_EncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ul←DataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

  Continues a multiple-part encryption operation.
- CK_RV C_EncryptFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

  Finishes a multiple-part encryption operation.
- CK_RV C_DecryptInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)

  Initialize decryption using the specified object.
- CK_RV C_Decrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ul←EncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

  Perform a single operation decryption in the given session.
- CK_RV C_DecryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pDataLen)

  Continues a multiple-part decryption operation.
- CK_RV C_DecryptFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG_PTR p←DataLen)

  Finishes a multiple-part decryption operation.
- CK_RV C_DigestInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism)

  Initializes a message-digesting operation using the specified mechanism in the specified session.
- CK_RV C_Digest (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Digest the specified data in a one-pass operation and return the resulting digest.
- CK_RV C_DigestUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part digesting operation.
- CK_RV C_DigestKey (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Update a running digest operation by digesting a secret key with the specified handle.
- CK_RV C_DigestFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Finishes a multiple-part digesting operation.
- CK_RV C_SignInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initialize a signing operation using the specified key and mechanism.

- CK_RV C_Sign (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Sign the data in a single pass operation.
- CK_RV C_SignUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part signature operation.
- CK_RV C_SignFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Finishes a multiple-part signature operation.
- CK_RV C_SignRecoverInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a signature operation, where the data can be recovered from the signature.
- CK_RV C_SignRecover (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Signs single-part data, where the data can be recovered from the signature.
- CK_RV C_VerifyInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation using the specified key and mechanism.
- CK_RV C_Verify (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Verifies a signature on single-part data.
- CK_RV C_VerifyUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part verification operation.
- CK_RV C_VerifyFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Finishes a multiple-part verification operation.
- CK_RV C_VerifyRecoverInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation where the data is recovered from the signature.
- CK_RV C_VerifyRecover (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

  Verifies a signature on single-part data, where the data is recovered from the signature.
- CK_RV C_DigestEncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pEncryptedPart, CK_ULONG_PTR pulEncryptedPartLen)

  Continues simultaneous multiple-part digesting and encryption operations.
- CK_RV C_DecryptDigestUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pDecryptedPart, CK_ULONG_PTR pulDecryptedPartLen)

  Continues simultaneous multiple-part decryption and digesting operations.
- CK_RV C_SignEncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pEncryptedPart, CK_ULONG_PTR pulEncryptedPartLen)

  Continues simultaneous multiple-part signature and encryption operations.
- CK_RV C_DecryptVerifyUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedPart, CK_ULONG ulEncryptedPartLen, CK_BYTE_PTR pPart, CK_ULONG_PTR pulPartLen)

  Continues simultaneous multiple-part decryption and verification operations.
- CK_RV C_GenerateKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

  Generates a secret key using the specified mechanism.
- CK_RV C_GenerateKeyPair (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR pPrivateKeyTemplate, CK_ULONG ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)

  Generates a public-key/private-key pair using the specified mechanism.
- CK_RV C_WrapKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hWrappingKey, CK_OBJECT_HANDLE hKey, CK_BYTE_PTR pWrappedKey, CK_ULONG_PTR pulWrappedKeyLen)

Wraps (encrypts) the specified key using the specified wrapping key and mechanism.

- CK_RV C_UnwrapKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hUnwrappingKey, CK_BYTE_PTR pWrappedKey, CK_ULONG ulWrappedKey↵ Len, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

  Unwraps (decrypts) the specified key using the specified unwrapping key.
- CK_RV C_DeriveKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hBaseKey, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

  Derive a key from the specified base key.
- CK_RV C_SeedRandom (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSeed, CK_ULONG ul↵ SeedLen)

  Mixes in additional seed material to the random number generator.
- CK_RV C_GenerateRandom (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pRandomData, CK_ULONG ulRandomLen)

  Generate the specified amount of random data.
- CK_RV C_GetFunctionStatus (CK_SESSION_HANDLE hSession)

  Legacy function - see PKCS#11 v2.40.
- CK_RV C_CancelFunction (CK_SESSION_HANDLE hSession)

  Legacy function.
- CK_RV C_WaitForSlotEvent (CK_FLAGS flags, CK_SLOT_ID_PTR pSlot, CK_VOID_PTR pReserved)

  Wait for a slot event (token insertion, removal, etc) on the specified slot to occur.
- CK_RV pkcs11_mech_get_list (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount)
- CK_RV pkcs_mech_get_info (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo)
- CK_RV pkcs11_object_alloc (pkcs11_object_ptr ∗ppObject)

  ∗∗
- CK_RV pkcs11_object_free (pkcs11_object_ptr pObject)
- CK_RV pkcs11_object_check (pkcs11_object_ptr ∗ppObject, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_object_get_handle (pkcs11_object_ptr pObject, CK_OBJECT_HANDLE_PTR phObject)
- CK_RV pkcs11_object_get_name (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_class (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_destroyable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_size (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ULONG_PTR pulSize)
- CK_RV pkcs11_object_find (pkcs11_object_ptr ∗ppObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_object_create (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phObject)

  Create a new object on the token in the specified session using the given attribute template.
- CK_RV pkcs11_object_destroy (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Destroy the specified object.
- CK_RV pkcs11_object_deinit (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_object_load_handle_info (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_object_is_private (pkcs11_object_ptr pObject, CK_BBOOL ∗is_private)

  Checks the attributes of the underlying cryptographic asset to determine if it is a private key - this changes the way the associated public key is referenced.
- CK_RV pkcs11_os_create_mutex (CK_VOID_PTR_PTR ppMutex)

  Application callback for creating a mutex object.
- CK_RV pkcs11_os_destroy_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_lock_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_unlock_mutex (CK_VOID_PTR pMutex)
- pkcs11_session_ctx_ptr pkcs11_get_session_context (CK_SESSION_HANDLE hSession)

- CK_RV pkcs11_session_check (pkcs11_session_ctx_ptr ∗pSession, CK_SESSION_HANDLE hSession)

  Check if the session is initialized properly.
- CK_RV pkcs11_session_open (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_NOTIFY notify, CK_SESSION_HANDLE_PTR phSession)
- CK_RV pkcs11_session_close (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_session_closeall (CK_SLOT_ID slotID)

  Close all sessions for a given slot - not actually all open sessions.
- CK_RV pkcs11_session_get_info (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo)

  Obtains information about a particular session.
- CK_RV pkcs11_session_login (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)
- CK_RV pkcs11_session_logout (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_signature_sign_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initialize a signing operation using the specified key and mechanism.
- CK_RV pkcs11_signature_sign (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Sign the data in a single pass operation.
- CK_RV pkcs11_signature_sign_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part signature operation.
- CK_RV pkcs11_signature_sign_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Finishes a multiple-part signature operation.
- CK_RV pkcs11_signature_verify_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation using the specified key and mechanism.
- CK_RV pkcs11_signature_verify (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Verifies a signature on single-part data.
- CK_RV pkcs11_signature_verify_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part verification operation.
- CK_RV pkcs11_signature_verify_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Finishes a multiple-part verification operation.
- pkcs11_slot_ctx_ptr pkcs11_slot_get_context (pkcs11_lib_ctx_ptr lib_ctx, CK_SLOT_ID slotID)

  Retrieve the current slot context.
- CK_VOID_PTR pkcs11_slot_initslots (CK_ULONG pulCount)
- CK_RV pkcs11_slot_config (CK_SLOT_ID slotID)
- CK_RV pkcs11_slot_init (CK_SLOT_ID slotID)
- CK_RV pkcs11_slot_get_list (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pulCount)
- CK_RV pkcs11_slot_get_info (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo)

  Obtains information about a particular slot.
- CK_RV pkcs11_token_init (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen, CK_UTF8CHAR_PTR pLabel)
- CK_RV pkcs11_token_get_access_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_writable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_storage (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_info (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo)

  Obtains information about a particular token.

- CK_RV pkcs11_token_random (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pRandomData, CK_ULONG ulRandomLen)

  Generate the specified amount of random data.
- CK_RV pkcs11_token_convert_pin_to_key (const CK_UTF8CHAR_PTR pPin, const CK_ULONG ulPinLen, const CK_UTF8CHAR_PTR pSalt, const CK_ULONG ulSaltLen, CK_BYTE_PTR pKey, CK_ULONG ulKeyLen)
- CK_RV pkcs11_token_set_pin (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pOldPin, CK_ULONG ulOldLen, CK_UTF8CHAR_PTR pNewPin, CK_ULONG ulNewLen)
- void pkcs11_util_escape_string (CK_UTF8CHAR_PTR buf, CK_ULONG buf_len)
- CK_RV pkcs11_util_convert_rv (ATCA_STATUS status)
- int pkcs11_util_memset (void ∗dest, size_t destsz, int ch, size_t count)

Variables

- const pkcs11_attrib_model pkcs11_cert_x509public_attributes [ ]
- const CK_ULONG pkcs11_cert_x509public_attributes_count = sizeof(pkcs11_cert_x509public_attributes) / sizeof(pkcs11_cert_x509public_attributes [0])
- const pkcs11_attrib_model pkcs11_cert_wtlspublic_attributes [ ]
- const CK_ULONG pkcs11_cert_wtlspublic_attributes_count = sizeof(pkcs11_cert_wtlspublic_attributes) / sizeof(pkcs11_cert_wtlspublic_attributes [0])
- const pkcs11_attrib_model pkcs11_cert_x509_attributes [ ]
- const CK_ULONG pkcs11_cert_x509_attributes_count = sizeof(pkcs11_cert_x509_attributes) / sizeof(pkcs11_cert_x509_attributes [0])
- const char pkcs11_lib_manufacturer_id [ ] = "Microchip Technology Inc"
- const char pkcs11_lib_description [ ] = "Cryptoauthlib PKCS11 Interface"
- const pkcs11_attrib_model pkcs11_key_public_attributes [ ]
- const CK_ULONG pkcs11_key_public_attributes_count = sizeof(pkcs11_key_public_attributes) / sizeof(pkcs11_key_public_attributes [0])
- const pkcs11_attrib_model pkcs11_key_ec_public_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_private_attributes [ ]
- const CK_ULONG pkcs11_key_private_attributes_count = sizeof(pkcs11_key_private_attributes) / sizeof(pkcs11_key_private_attributes [0])
- const pkcs11_attrib_model pkcs11_key_rsa_private_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_ec_private_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_secret_attributes [ ]
- const CK_ULONG pkcs11_key_secret_attributes_count = sizeof(pkcs11_key_secret_attributes) / sizeof(pkcs11_key_secret_attributes [0])
- pkcs11_object_cache_t pkcs11_object_cache [PKCS11_MAX_OBJECTS_ALLOWED]
- const pkcs11_attrib_model pkcs11_object_monotonic_attributes [ ]
- const CK_ULONG pkcs11_object_monotonic_attributes_count = sizeof(pkcs11_object_monotonic_attributes) / sizeof(pkcs11_object_monotonic_attributes [0])

8.12.1 Detailed Description

8.12.2 Macro Definition Documentation

8.12.2.1 PCKS11_MECH_ECC508_EC_CAPABILITY

```
#define PCKS11_MECH_ECC508_EC_CAPABILITY (CKF_EC_F_P | CKF_EC_NAMEDCURVE | CKF_EC_UNCOMPRESS)
```

8.12.2.2 TABLE_SIZE

```
#define TABLE_SIZE(
             x ) sizeof(x) / sizeof(x[0])
```

8.12.3 Typedef Documentation

8.12.3.1 pcks11_mech_table_e

```
typedef struct _pcks11_mech_table_e pcks11_mech_table_e
```

8.12.3.2 pcks11_mech_table_ptr

```
typedef struct _pcks11_mech_table_e * pcks11_mech_table_ptr
```

8.12.4 Function Documentation

8.12.4.1 C_CancelFunction()

```
CK_RV C_CancelFunction (
             CK_SESSION_HANDLE hSession )
```

Legacy function.

8.12.4.2 C_CloseAllSessions()

```
CK_RV C_CloseAllSessions (
             CK_SLOT_ID slotID )
```

Close all open sessions.

8.12.4.3 C_CloseSession()

```
CK_RV C_CloseSession (
            CK_SESSION_HANDLE hSession )
```

Close the given session.

8.12.4.4 C_CopyObject()

```
CK_RV C_CopyObject (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phNewObject )
```

Create a copy of the object with the specified handle.

8.12.4.5 C_CreateObject()

```
CK_RV C_CreateObject (
            CK_SESSION_HANDLE hSession,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phObject )
```

Create a new object on the token in the specified session using the given attribute template.

8.12.4.6 C_Decrypt()

```
CK_RV C_Decrypt (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG ulEncryptedDataLen,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pulDataLen )
```

Perform a single operation decryption in the given session.

8.12.4.7 C_DecryptDigestUpdate()

```
CK_RV C_DecryptDigestUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen,
            CK_BYTE_PTR pDecryptedPart,
            CK_ULONG_PTR pulDecryptedPartLen )
```

Continues simultaneous multiple-part decryption and digesting operations.

8.12.4.8 C_DecryptFinal()

```
CK_RV C_DecryptFinal (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pDataLen )
```

Finishes a multiple-part decryption operation.

8.12.4.9 C_DecryptInit()

```
CK_RV C_DecryptInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hObject )
```

Initialize decryption using the specified object.

8.12.4.10 C_DecryptUpdate()

```
CK_RV C_DecryptUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG ulEncryptedDataLen,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pDataLen )
```

Continues a multiple-part decryption operation.

8.12.4.11 C_DecryptVerifyUpdate()

```
CK_RV C_DecryptVerifyUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedPart,
            CK_ULONG ulEncryptedPartLen,
            CK_BYTE_PTR pPart,
            CK_ULONG_PTR pulPartLen )
```

Continues simultaneous multiple-part decryption and verification operations.

8.12.4.12 C_DeriveKey()

```
CK_RV C_DeriveKey (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hBaseKey,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phKey )
```

Derive a key from the specified base key.

8.12.4.13 C_DestroyObject()

```
CK_RV C_DestroyObject (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject )
```

Destroy the specified object.

8.12.4.14 C_Digest()

```
CK_RV C_Digest (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Digest the specified data in a one-pass operation and return the resulting digest.

8.12.4.15 C_DigestEncryptUpdate()

```
CK_RV C_DigestEncryptUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen,
            CK_BYTE_PTR pEncryptedPart,
            CK_ULONG_PTR pulEncryptedPartLen )
```

Continues simultaneous multiple-part digesting and encryption operations.

8.12.4.16 C_DigestFinal()

```
CK_RV C_DigestFinal (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Finishes a multiple-part digesting operation.

8.12.4.17 C_DigestInit()

```
CK_RV C_DigestInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism )
```

Initializes a message-digesting operation using the specified mechanism in the specified session.

8.12.4.18 C_DigestKey()

```
CK_RV C_DigestKey (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject )
```

Update a running digest operation by digesting a secret key with the specified handle.

8.12.4.19 C_DigestUpdate()

```
CK_RV C_DigestUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part digesting operation.

8.12.4.20 C_Encrypt()

```
CK_RV C_Encrypt (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

Perform a single operation encryption operation in the specified session.

8.12.4.21 C_EncryptFinal()

```
CK_RV C_EncryptFinal (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

Finishes a multiple-part encryption operation.

8.12.4.22 C_EncryptInit()

```
CK_RV C_EncryptInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hObject )
```

Initializes an encryption operation using the specified mechanism and session.

8.12.4.23 C_EncryptUpdate()

```
CK_RV C_EncryptUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

Continues a multiple-part encryption operation.

8.12.4.24 C_Finalize()

```
CK_RV C_Finalize (
            CK_VOID_PTR pReserved )
```

Clean up miscellaneous Cryptoki-associated resources.

8.12.4.25 C_FindObjects()

```
CK_RV C_FindObjects (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE_PTR phObject,
            CK_ULONG ulMaxObjectCount,
            CK_ULONG_PTR pulObjectCount )
```

Continue the search for objects in the specified session.

8.12.4.26 C_FindObjectsFinal()

```
CK_RV C_FindObjectsFinal (
            CK_SESSION_HANDLE hSession )
```

Finishes an object search operation (and cleans up)

8.12.4.27 C_FindObjectsInit()

```
CK_RV C_FindObjectsInit (
            CK_SESSION_HANDLE hSession,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

Initializes an object search in the specified session using the specified attribute template as search parameters.

8.12.4.28 C_GenerateKey()

```
CK_RV C_GenerateKey (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phKey )
```

Generates a secret key using the specified mechanism.

8.12.4.29 C_GenerateKeyPair()

```
CK_RV C_GenerateKeyPair (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_ATTRIBUTE_PTR pPublicKeyTemplate,
            CK_ULONG ulPublicKeyAttributeCount,
            CK_ATTRIBUTE_PTR pPrivateKeyTemplate,
            CK_ULONG ulPrivateKeyAttributeCount,
            CK_OBJECT_HANDLE_PTR phPublicKey,
            CK_OBJECT_HANDLE_PTR phPrivateKey )
```

Generates a public-key/private-key pair using the specified mechanism.

8.12.4.30 C_GenerateRandom()

```
CK_RV C_GenerateRandom (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pRandomData,
            CK_ULONG ulRandomLen )
```

Generate the specified amount of random data.

8.12.4.31 C_GetAttributeValue()

```
CK_RV C_GetAttributeValue (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

Obtains an attribute value of an object.

8.12.4.32 C_GetFunctionList()

```
CK_RV C_GetFunctionList (
            CK_FUNCTION_LIST_PTR_PTR ppFunctionList )
```

Obtains entry points of Cryptoki library functions.

8.12.4.33 C_GetFunctionStatus()

```
CK_RV C_GetFunctionStatus (
            CK_SESSION_HANDLE hSession )
```

Legacy function - see PKCS#11 v2.40.

8.12.4.34 C_GetInfo()

```
CK_RV C_GetInfo (
            CK_INFO_PTR pInfo )
```

Obtains general information about Cryptoki.

8.12.4.35 C_GetMechanismInfo()

```
CK_RV C_GetMechanismInfo (
            CK_SLOT_ID slotID,
            CK_MECHANISM_TYPE type,
            CK_MECHANISM_INFO_PTR pInfo )
```

Obtains information about a particular mechanism of a token (in a slot)

8.12.4.36 C_GetMechanismList()

```
CK_RV C_GetMechanismList (
            CK_SLOT_ID slotID,
            CK_MECHANISM_TYPE_PTR pMechanismList,
            CK_ULONG_PTR pulCount )
```

Obtains a list of mechanisms supported by a token (in a slot)

8.12.4.37 C_GetObjectSize()

```
CK_RV C_GetObjectSize (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ULONG_PTR pulSize )
```

Obtains the size of an object in bytes.

8.12.4.38 C_GetOperationState()

```
CK_RV C_GetOperationState (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pOperationState,
            CK_ULONG_PTR pulOperationStateLen )
```

Obtains the cryptographic operations state of a session.

8.12.4.39 C_GetSessionInfo()

```
CK_RV C_GetSessionInfo (
            CK_SESSION_HANDLE hSession,
            CK_SESSION_INFO_PTR pInfo )
```

Retrieve information about the specified session.

8.12.4.40 C_GetSlotInfo()

```
CK_RV C_GetSlotInfo (
            CK_SLOT_ID slotID,
            CK_SLOT_INFO_PTR pInfo )
```

Obtains information about a particular slot.

8.12.4.41 C_GetSlotList()

```
CK_RV C_GetSlotList (
            CK_BBOOL tokenPresent,
            CK_SLOT_ID_PTR pSlotList,
            CK_ULONG_PTR pulCount )
```

Obtains a list of slots in the system.

8.12.4.42 C_GetTokenInfo()

```
CK_RV C_GetTokenInfo (
            CK_SLOT_ID slotID,
            CK_TOKEN_INFO_PTR pInfo )
```

Obtains information about a particular token.

8.12.4.43 C_Initialize()

```
CK_RV C_Initialize (
            CK_VOID_PTR pInitArgs )
```

Initializes Cryptoki library NOTES: If pInitArgs is a non-NULL_PTR is must dereference to a CK_C_INITIALIZE_ARGS structure.

8.12.4.44 C_InitPIN()

```
CK_RV C_InitPIN (
            CK_SESSION_HANDLE hSession,
            CK_UTF8CHAR_PTR pPin,
            CK_ULONG ulPinLen )
```

Initializes the normal user's PIN.

8.12.4.45 C_InitToken()

```
CK_RV C_InitToken (
            CK_SLOT_ID slotID,
            CK_UTF8CHAR_PTR pPin,
            CK_ULONG ulPinLen,
            CK_UTF8CHAR_PTR pLabel )
```

Initializes a token (in a slot)

8.12.4.46 C_Login()

```
CK_RV C_Login (
            CK_SESSION_HANDLE hSession,
            CK_USER_TYPE userType,
            CK_UTF8CHAR_PTR pPin,
            CK_ULONG ulPinLen )
```

Login on the token in the specified session.

8.12.4.47 C_Logout()

```
CK_RV C_Logout (
            CK_SESSION_HANDLE hSession )
```

Log out of the token in the specified session.

8.12.4.48 C_OpenSession()

```
CK_RV C_OpenSession (
            CK_SLOT_ID slotID,
            CK_FLAGS flags,
            CK_VOID_PTR pApplication,
            CK_NOTIFY notify,
            CK_SESSION_HANDLE_PTR phSession )
```

Opens a connection between an application and a particular token or sets up an application callback for token insertion.

8.12.4.49 C_SeedRandom()

```
CK_RV C_SeedRandom (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSeed,
            CK_ULONG ulSeedLen )
```

Mixes in additional seed material to the random number generator.

8.12.4.50 C_SetAttributeValue()

```
CK_RV C_SetAttributeValue (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

Change or set the value of the specified attributes on the specified object.

8.12.4.51 C_SetOperationState()

```
CK_RV C_SetOperationState (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pOperationState,
            CK_ULONG ulOperationStateLen,
            CK_OBJECT_HANDLE hEncryptionKey,
            CK_OBJECT_HANDLE hAuthenticationKey )
```

Sets the cryptographic operations state of a session.

8.12.4.52 C_SetPIN()

```
CK_RV C_SetPIN (
            CK_SESSION_HANDLE hSession,
            CK_UTF8CHAR_PTR pOldPin,
            CK_ULONG ulOldLen,
            CK_UTF8CHAR_PTR pNewPin,
            CK_ULONG ulNewLen )
```

Modifies the PIN of the current user.

8.12.4.53 C_Sign()

```
CK_RV C_Sign (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pSignature,
            CK_ULONG_PTR pulSignatureLen )
```

Sign the data in a single pass operation.

8.12.4.54 C_SignEncryptUpdate()

```
CK_RV C_SignEncryptUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen,
            CK_BYTE_PTR pEncryptedPart,
            CK_ULONG_PTR pulEncryptedPartLen )
```

Continues simultaneous multiple-part signature and encryption operations.

8.12.4.55 C_SignFinal()

```
CK_RV C_SignFinal (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSignature,
            CK_ULONG_PTR pulSignatureLen )
```

Finishes a multiple-part signature operation.

8.12.4.56 C_SignInit()

```
CK_RV C_SignInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initialize a signing operation using the specified key and mechanism.

8.12.4.57 C_SignRecover()

```
CK_RV C_SignRecover (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pSignature,
            CK_ULONG_PTR pulSignatureLen )
```

Signs single-part data, where the data can be recovered from the signature.

8.12.4.58 C_SignRecoverInit()

```
CK_RV C_SignRecoverInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initializes a signature operation, where the data can be recovered from the signature.

8.12.4.59 C_SignUpdate()

```
CK_RV C_SignUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part signature operation.

8.12.4.60 C_UnwrapKey()

```
CK_RV C_UnwrapKey (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hUnwrappingKey,
            CK_BYTE_PTR pWrappedKey,
            CK_ULONG ulWrappedKeyLen,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phKey )
```

Unwraps (decrypts) the specified key using the specified unwrapping key.

8.12.4.61 C_Verify()

```
CK_RV C_Verify (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pSignature,
            CK_ULONG ulSignatureLen )
```

Verifies a signature on single-part data.

8.12.4.62 C_VerifyFinal()

```
CK_RV C_VerifyFinal (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSignature,
            CK_ULONG ulSignatureLen )
```

Finishes a multiple-part verification operation.

8.12.4.63 C_VerifyInit()

```
CK_RV C_VerifyInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initializes a verification operation using the specified key and mechanism.

8.12.4.64 C_VerifyRecover()

```
CK_RV C_VerifyRecover (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSignature,
            CK_ULONG ulSignatureLen,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pulDataLen )
```

Verifies a signature on single-part data, where the data is recovered from the signature.

8.12.4.65 C_VerifyRecoverInit()

```
CK_RV C_VerifyRecoverInit (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initializes a verification operation where the data is recovered from the signature.

8.12.4.66 C_VerifyUpdate()

```
CK_RV C_VerifyUpdate (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part verification operation.

8.12.4.67 C_WaitForSlotEvent()

```
CK_RV C_WaitForSlotEvent (
            CK_FLAGS flags,
            CK_SLOT_ID_PTR pSlot,
            CK_VOID_PTR pReserved )
```

Wait for a slot event (token insertion, removal, etc) on the specified slot to occur.

8.12.4.68 C_WrapKey()

```
CK_RV C_WrapKey (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hWrappingKey,
            CK_OBJECT_HANDLE hKey,
            CK_BYTE_PTR pWrappedKey,
            CK_ULONG_PTR pulWrappedKeyLen )
```

Wraps (encrypts) the specified key using the specified wrapping key and mechanism.

8.12.4.69 pkcs11_attrib_empty()

```
CK_RV pkcs11_attrib_empty (
            const CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.70 pkcs11_attrib_false()

```
CK_RV pkcs11_attrib_false (
            const CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.71 pkcs11_attrib_fill()

```
CK_RV pkcs11_attrib_fill (
            CK_ATTRIBUTE_PTR pAttribute,
            const CK_VOID_PTR pData,
            const CK_ULONG ulSize )
```

Perform the nessasary checks and copy data into an attribute structure.

The ulValueLen field is modified to hold the exact length of the specified attribute for the object. In the special case of an attribute whose value is an array of attributes, for example CKA_WRAP_TEMPLATE, where it is passed in with pValue not NULL, then if the pValue of elements within the array is NULL_PTR then the ulValueLen of elements within the array will be set to the required length. If the pValue of elements within the array is not NULL_PTR, then the ulValueLen element of attributes within the array MUST reflect the space that the corresponding pValue points to, and pValue is filled in if there is sufficient room.Therefore it is important to initialize the contents of a buffer before calling C_GetAttributeValue to get such an array value. If any ulValueLen within the array isn't large enough, it will be set to CK_UNAVAILABLE_INFORMATION and the function will return CKR_BUFFER_TOO_SMALL, as it does if an attribute in the pTemplate argument has ulValueLen too small Note that any attribute whose value is an array of attributes is identifiable by virtue of the attribute type having the CKF_ARRAY_ATTRIBUTE bit set.

8.12.4.72 pkcs11_attrib_true()

```
CK_RV pkcs11_attrib_true (
            const CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.73 pkcs11_attrib_value()

```
CK_RV pkcs11_attrib_value (
            CK_ATTRIBUTE_PTR pAttribute,
            const CK_ULONG ulValue,
            const CK_ULONG ulSize )
```

Helper function to write a numerical value to an attribute buffer.

8.12.4.74 pkcs11_cert_get_authority_key_id()

```
CK_RV pkcs11_cert_get_authority_key_id (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.75 pkcs11_cert_get_encoded()

```
CK_RV pkcs11_cert_get_encoded (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.76 pkcs11_cert_get_subject()

```
CK_RV pkcs11_cert_get_subject (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.77 pkcs11_cert_get_subject_key_id()

```
CK_RV pkcs11_cert_get_subject_key_id (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.78 pkcs11_cert_get_trusted_flag()

```
CK_RV pkcs11_cert_get_trusted_flag (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.79 pkcs11_cert_get_type()

```
CK_RV pkcs11_cert_get_type (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.80 pkcs11_cert_x509_write()

```
CK_RV pkcs11_cert_x509_write (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.81 pkcs11_config_cert()

```
CK_RV pkcs11_config_cert (
            pkcs11_lib_ctx_ptr pLibCtx,
            pkcs11_slot_ctx_ptr pSlot,
            pkcs11_object_ptr pObject,
            CK_ATTRIBUTE_PTR pLabel )
```

8.12.4.82 pkcs11_config_init_cert()

```
void pkcs11_config_init_cert (
            pkcs11_object_ptr pObject,
            char * label,
            size_t len )
```

8.12.4.83 pkcs11_config_init_private()

```
void pkcs11_config_init_private (
            pkcs11_object_ptr pObject,
            char * label,
            size_t len )
```

8.12.4.84 pkcs11_config_init_public()

```
void pkcs11_config_init_public (
            pkcs11_object_ptr pObject,
            char * label,
            size_t len )
```

8.12.4.85 pkcs11_config_init_secret()

```
void pkcs11_config_init_secret (
            pkcs11_object_ptr pObject,
            char * label,
            size_t len,
            uint8_t keylen )
```

8.12.4.86 pkcs11_config_key()

```
CK_RV pkcs11_config_key (
            pkcs11_lib_ctx_ptr pLibCtx,
            pkcs11_slot_ctx_ptr pSlot,
            pkcs11_object_ptr pObject,
            CK_ATTRIBUTE_PTR pLabel )
```

8.12.4.87 pkcs11_config_load()

```
CK_RV pkcs11_config_load (
            pkcs11_slot_ctx_ptr slot_ctx )
```

8.12.4.88 pkcs11_config_load_objects()

```
CK_RV pkcs11_config_load_objects (
            pkcs11_slot_ctx_ptr slot_ctx )
```

8.12.4.89 pkcs11_config_remove_object()

```
CK_RV pkcs11_config_remove_object (
            pkcs11_lib_ctx_ptr pLibCtx,
            pkcs11_slot_ctx_ptr pSlot,
            pkcs11_object_ptr pObject )
```

8.12.4.90 pkcs11_decrypt()

```
CK_RV pkcs11_decrypt (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG ulEncryptedDataLen,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pulDataLen )
```

8.12.4.91 pkcs11_decrypt_final()

```
CK_RV pkcs11_decrypt_final (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pulDataLen )
```

Finishes a multiple-part decryption operation.

8.12.4.92 pkcs11_decrypt_init()

```
CK_RV pkcs11_decrypt_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hObject )
```

8.12.4.93 pkcs11_decrypt_update()

```
CK_RV pkcs11_decrypt_update (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG ulEncryptedDataLen,
            CK_BYTE_PTR pData,
            CK_ULONG_PTR pulDataLen )
```

8.12.4.94 pkcs11_deinit()

```
CK_RV pkcs11_deinit (
            CK_VOID_PTR pReserved )
```

8.12.4.95 pkcs11_encrypt()

```
CK_RV pkcs11_encrypt (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

8.12.4.96 pkcs11_encrypt_final()

```
CK_RV pkcs11_encrypt_final (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

Finishes a multiple-part encryption operation.

8.12.4.97 pkcs11_encrypt_init()

```
CK_RV pkcs11_encrypt_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hObject )
```

8.12.4.98 pkcs11_encrypt_update()

```
CK_RV pkcs11_encrypt_update (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pEncryptedData,
            CK_ULONG_PTR pulEncryptedDataLen )
```

8.12.4.99 pkcs11_find_continue()

```
CK_RV pkcs11_find_continue (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE_PTR phObject,
            CK_ULONG ulMaxObjectCount,
            CK_ULONG_PTR pulObjectCount )
```

8.12.4.100 pkcs11_find_finish()

```
CK_RV pkcs11_find_finish (
            CK_SESSION_HANDLE hSession )
```

8.12.4.101 pkcs11_find_get_attribute()

```
CK_RV pkcs11_find_get_attribute (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

8.12.4.102 pkcs11_find_init()

```
CK_RV pkcs11_find_init (
            CK_SESSION_HANDLE hSession,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

8.12.4.103 pkcs11_get_context()

```
pkcs11_lib_ctx_ptr pkcs11_get_context (
            void )
```

Retrieve the current library context.

8.12.4.104 pkcs11_get_lib_info()

```
CK_RV pkcs11_get_lib_info (
            CK_INFO_PTR pInfo )
```

Obtains general information about Cryptoki.

8.12.4.105 pkcs11_get_session_context()

```
pkcs11_session_ctx_ptr pkcs11_get_session_context (
            CK_SESSION_HANDLE hSession )
```

8.12.4.106 pkcs11_init()

```
CK_RV pkcs11_init (
            CK_C_INITIALIZE_ARGS_PTR pInitArgs )
```

Initializes the PKCS11 API Library for Cryptoauthlib.

8.12.4.107 pkcs11_init_check()

```
CK_RV pkcs11_init_check (
            pkcs11_lib_ctx_ptr * ppContext,
            CK_BBOOL lock )
```

Check if the library is initialized properly.

8.12.4.108 pkcs11_key_derive()

```
CK_RV pkcs11_key_derive (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hBaseKey,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phKey )
```

8.12.4.109 pkcs11_key_generate()

```
CK_RV pkcs11_key_generate (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phKey )
```

8.12.4.110 pkcs11_key_generate_pair()

```
CK_RV pkcs11_key_generate_pair (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_ATTRIBUTE_PTR pPublicKeyTemplate,
            CK_ULONG ulPublicKeyAttributeCount,
            CK_ATTRIBUTE_PTR pPrivateKeyTemplate,
            CK_ULONG ulPrivateKeyAttributeCount,
            CK_OBJECT_HANDLE_PTR phPublicKey,
            CK_OBJECT_HANDLE_PTR phPrivateKey )
```

8.12.4.111 pkcs11_key_write()

```
CK_RV pkcs11_key_write (
            CK_VOID_PTR pSession,
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.112 pkcs11_lock_context()

```
CK_RV pkcs11_lock_context (
            pkcs11_lib_ctx_ptr pContext )
```

8.12.4.113 pkcs11_mech_get_list()

```
CK_RV pkcs11_mech_get_list (
            CK_SLOT_ID slotID,
            CK_MECHANISM_TYPE_PTR pMechanismList,
            CK_ULONG_PTR pulCount )
```

8.12.4.114 pkcs11_object_alloc()

```
CK_RV pkcs11_object_alloc (
            pkcs11_object_ptr * ppObject )
```

**

**

8.12.4.115 pkcs11_object_check()

```
CK_RV pkcs11_object_check (
            pkcs11_object_ptr * ppObject,
            CK_OBJECT_HANDLE hObject )
```

8.12.4.116 pkcs11_object_create()

```
CK_RV pkcs11_object_create (
            CK_SESSION_HANDLE hSession,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount,
            CK_OBJECT_HANDLE_PTR phObject )
```

Create a new object on the token in the specified session using the given attribute template.

8.12.4.117 pkcs11_object_deinit()

```
CK_RV pkcs11_object_deinit (
            pkcs11_lib_ctx_ptr pContext )
```

8.12.4.118 pkcs11_object_destroy()

```
CK_RV pkcs11_object_destroy (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject )
```

Destroy the specified object.

8.12.4.119 pkcs11_object_find()

```
CK_RV pkcs11_object_find (
            pkcs11_object_ptr * ppObject,
            CK_ATTRIBUTE_PTR pTemplate,
            CK_ULONG ulCount )
```

8.12.4.120 pkcs11_object_free()

```
CK_RV pkcs11_object_free (
            pkcs11_object_ptr pObject )
```

8.12.4.121 pkcs11_object_get_class()

```
CK_RV pkcs11_object_get_class (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.122 pkcs11_object_get_destroyable()

```
CK_RV pkcs11_object_get_destroyable (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.123 pkcs11_object_get_handle()

```
CK_RV pkcs11_object_get_handle (
            pkcs11_object_ptr pObject,
            CK_OBJECT_HANDLE_PTR phObject )
```

8.12.4.124 pkcs11_object_get_name()

```
CK_RV pkcs11_object_get_name (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.125 pkcs11_object_get_size()

```
CK_RV pkcs11_object_get_size (
            CK_SESSION_HANDLE hSession,
            CK_OBJECT_HANDLE hObject,
            CK_ULONG_PTR pulSize )
```

8.12.4.126 pkcs11_object_get_type()

```
CK_RV pkcs11_object_get_type (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.127 pkcs11_object_is_private()

```
CK_RV pkcs11_object_is_private (
            pkcs11_object_ptr pObject,
            CK_BBOOL * is_private )
```

Checks the attributes of the underlying cryptographic asset to determine if it is a private key - this changes the way the associated public key is referenced.

8.12.4.128 pkcs11_object_load_handle_info()

```
CK_RV pkcs11_object_load_handle_info (
            pkcs11_lib_ctx_ptr pContext )
```

8.12.4.129 pkcs11_os_create_mutex()

```
CK_RV pkcs11_os_create_mutex (
            CK_VOID_PTR_PTR ppMutex )
```

Application callback for creating a mutex object.

Parameters

in,out	*ppMutex*	location to receive ptr to mutex

8.12.4.130 pkcs11_os_destroy_mutex()

```
CK_RV pkcs11_os_destroy_mutex (
            CK_VOID_PTR pMutex )
```

8.12.4.131 pkcs11_os_lock_mutex()

```
CK_RV pkcs11_os_lock_mutex (
            CK_VOID_PTR pMutex )
```

8.12.4.132 pkcs11_os_unlock_mutex()

```
CK_RV pkcs11_os_unlock_mutex (
            CK_VOID_PTR pMutex )
```

8.12.4.133 pkcs11_session_check()

```
CK_RV pkcs11_session_check (
            pkcs11_session_ctx_ptr * pSession,
            CK_SESSION_HANDLE hSession )
```

Check if the session is initialized properly.

8.12.4.134 pkcs11_session_close()

```
CK_RV pkcs11_session_close (
            CK_SESSION_HANDLE hSession )
```

8.12.4.135 pkcs11_session_closeall()

```
CK_RV pkcs11_session_closeall (
            CK_SLOT_ID slotID )
```

Close all sessions for a given slot - not actually all open sessions.

8.12.4.136 pkcs11_session_get_info()

```
CK_RV pkcs11_session_get_info (
            CK_SESSION_HANDLE hSession,
            CK_SESSION_INFO_PTR pInfo )
```

Obtains information about a particular session.

8.12.4.137 pkcs11_session_login()

```
CK_RV pkcs11_session_login (
            CK_SESSION_HANDLE hSession,
            CK_USER_TYPE userType,
            CK_UTF8CHAR_PTR pPin,
            CK_ULONG ulPinLen )
```

8.12.4.138 pkcs11_session_logout()

```
CK_RV pkcs11_session_logout (
            CK_SESSION_HANDLE hSession )
```

8.12.4.139 pkcs11_session_open()

```
CK_RV pkcs11_session_open (
            CK_SLOT_ID slotID,
            CK_FLAGS flags,
            CK_VOID_PTR pApplication,
            CK_NOTIFY notify,
            CK_SESSION_HANDLE_PTR phSession )
```

8.12.4.140 pkcs11_signature_sign()

```
CK_RV pkcs11_signature_sign (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pSignature,
            CK_ULONG_PTR pulSignatureLen )
```

Sign the data in a single pass operation.

8.12.4.141 pkcs11_signature_sign_continue()

```
CK_RV pkcs11_signature_sign_continue (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part signature operation.

8.12.4.142 pkcs11_signature_sign_finish()

```
CK_RV pkcs11_signature_sign_finish (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSignature,
            CK_ULONG_PTR pulSignatureLen )
```

Finishes a multiple-part signature operation.

8.12.4.143 pkcs11_signature_sign_init()

```
CK_RV pkcs11_signature_sign_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initialize a signing operation using the specified key and mechanism.

8.12.4.144 pkcs11_signature_verify()

```
CK_RV pkcs11_signature_verify (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pSignature,
            CK_ULONG ulSignatureLen )
```

Verifies a signature on single-part data.

8.12.4.145 pkcs11_signature_verify_continue()

```
CK_RV pkcs11_signature_verify_continue (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part verification operation.

8.12.4.146 pkcs11_signature_verify_finish()

```
CK_RV pkcs11_signature_verify_finish (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pSignature,
            CK_ULONG ulSignatureLen )
```

Finishes a multiple-part verification operation.

8.12.4.147 pkcs11_signature_verify_init()

```
CK_RV pkcs11_signature_verify_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism,
            CK_OBJECT_HANDLE hKey )
```

Initializes a verification operation using the specified key and mechanism.

8.12.4.148 pkcs11_slot_config()

```
CK_RV pkcs11_slot_config (
            CK_SLOT_ID slotID )
```

8.12.4.149 pkcs11_slot_get_context()

```
pkcs11_slot_ctx_ptr pkcs11_slot_get_context (
            pkcs11_lib_ctx_ptr lib_ctx,
            CK_SLOT_ID slotID )
```

Retrieve the current slot context.

8.12.4.150 pkcs11_slot_get_info()

```
CK_RV pkcs11_slot_get_info (
            CK_SLOT_ID slotID,
            CK_SLOT_INFO_PTR pInfo )
```

Obtains information about a particular slot.

8.12.4.151 pkcs11_slot_get_list()

```
CK_RV pkcs11_slot_get_list (
            CK_BBOOL tokenPresent,
            CK_SLOT_ID_PTR pSlotList,
            CK_ULONG_PTR pulCount )
```

8.12.4.152 pkcs11_slot_init()

```
CK_RV pkcs11_slot_init (
            CK_SLOT_ID slotID )
```

8.12.4.153 pkcs11_slot_initslots()

```
CK_VOID_PTR pkcs11_slot_initslots (
            CK_ULONG pulCount )
```

8.12.4.154 pkcs11_token_convert_pin_to_key()

```
CK_RV pkcs11_token_convert_pin_to_key (
            const CK_UTF8CHAR_PTR pPin,
            const CK_ULONG ulPinLen,
            const CK_UTF8CHAR_PTR pSalt,
            const CK_ULONG ulSaltLen,
            CK_BYTE_PTR pKey,
            CK_ULONG ulKeyLen )
```

8.12.4.155 pkcs11_token_get_access_type()

```
CK_RV pkcs11_token_get_access_type (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.156 pkcs11_token_get_info()

```
CK_RV pkcs11_token_get_info (
            CK_SLOT_ID slotID,
            CK_TOKEN_INFO_PTR pInfo )
```

Obtains information about a particular token.

8.12.4.157 pkcs11_token_get_storage()

```
CK_RV pkcs11_token_get_storage (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.158 pkcs11_token_get_writable()

```
CK_RV pkcs11_token_get_writable (
            CK_VOID_PTR pObject,
            CK_ATTRIBUTE_PTR pAttribute )
```

8.12.4.159 pkcs11_token_init()

```
CK_RV pkcs11_token_init (
            CK_SLOT_ID slotID,
            CK_UTF8CHAR_PTR pPin,
            CK_ULONG ulPinLen,
            CK_UTF8CHAR_PTR pLabel )
```

Write the configuration into the device and generate new keys

8.12.4.160 pkcs11_token_random()

```
CK_RV pkcs11_token_random (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pRandomData,
            CK_ULONG ulRandomLen )
```

Generate the specified amount of random data.

8.12.4.161 pkcs11_token_set_pin()

```
CK_RV pkcs11_token_set_pin (
            CK_SESSION_HANDLE hSession,
            CK_UTF8CHAR_PTR pOldPin,
            CK_ULONG ulOldLen,
            CK_UTF8CHAR_PTR pNewPin,
            CK_ULONG ulNewLen )
```

8.12.4.162 pkcs11_unlock_context()

```
CK_RV pkcs11_unlock_context (
            pkcs11_lib_ctx_ptr pContext )
```

8.12.4.163 pkcs11_util_convert_rv()

```
CK_RV pkcs11_util_convert_rv (
            ATCA_STATUS status )
```

8.12.4.164 pkcs11_util_escape_string()

```
void pkcs11_util_escape_string (
            CK_UTF8CHAR_PTR buf,
            CK_ULONG buf_len )
```

8.12.4.165 pkcs11_util_memset()

```
int pkcs11_util_memset (
            void * dest,
            size_t destsz,
            int ch,
            size_t count )
```

8.12.4.166 pkcs_mech_get_info()

```
CK_RV pkcs_mech_get_info (
            CK_SLOT_ID slotID,
            CK_MECHANISM_TYPE type,
            CK_MECHANISM_INFO_PTR pInfo )
```

8.12.5 Variable Documentation

8.12.5.1 pkcs11_cert_wtlspublic_attributes

```
const pkcs11_attrib_model pkcs11_cert_wtlspublic_attributes[ ]
```

CKO_CERTIFICATE (Type: CKC_WTLS) - WTLS Public Key Certificate Model

8.12.5.2 pkcs11_cert_wtlspublic_attributes_count

```
const CK_ULONG pkcs11_cert_wtlspublic_attributes_count = sizeof( pkcs11_cert_wtlspublic_attributes
) / sizeof( pkcs11_cert_wtlspublic_attributes [0])
```

8.12.5.3 pkcs11_cert_x509_attributes

```
const pkcs11_attrib_model pkcs11_cert_x509_attributes[ ]
```

CKO_CERTIFICATE (Type: CKC_X_509_ATTR_CERT) - X509 Attribute Certificate Model

8.12.5.4 pkcs11_cert_x509_attributes_count

```
const CK_ULONG pkcs11_cert_x509_attributes_count = sizeof( pkcs11_cert_x509_attributes ) /
sizeof( pkcs11_cert_x509_attributes [0])
```

8.12.5.5 pkcs11_cert_x509public_attributes

```
const pkcs11_attrib_model pkcs11_cert_x509public_attributes[ ]
```

CKO_CERTIFICATE (Type: CKC_X_509) - X509 Public Key Certificate Model

8.12.5.6 pkcs11_cert_x509public_attributes_count

```
const CK_ULONG pkcs11_cert_x509public_attributes_count = sizeof( pkcs11_cert_x509public_attributes
) / sizeof( pkcs11_cert_x509public_attributes [0])
```

8.12.5.7 pkcs11_key_ec_private_attributes

```
const pkcs11_attrib_model pkcs11_key_ec_private_attributes[ ]
```

Initial value:
```
= {
    { 0x00000180UL , pkcs11_key_get_ec_params },
    { 0x00000181UL , pkcs11_key_get_ec_point },
}
```

CKO_PRIVATE_KEY (Type: CKK_EC) - EC/ECDSA Public Key Object Model

8.12.5.8 pkcs11_key_ec_public_attributes

```
const pkcs11_attrib_model pkcs11_key_ec_public_attributes[ ]
```

Initial value:
```
= {
    { 0x00000180UL , pkcs11_key_get_ec_params },
    { 0x00000181UL , pkcs11_key_get_ec_point },
}
```

CKO_PUBLIC_KEY (Type: CKK_EC) - EC/ECDSA Public Key Object Model

8.12.5.9 pkcs11_key_private_attributes

```
const pkcs11_attrib_model pkcs11_key_private_attributes[ ]
```

CKO_PRIVATE_KEY - Private Key Object Base Model

8.12.5.10 pkcs11_key_private_attributes_count

```
const CK_ULONG pkcs11_key_private_attributes_count = sizeof( pkcs11_key_private_attributes ) /
sizeof( pkcs11_key_private_attributes [0])
```

8.12.5.11 pkcs11_key_public_attributes

```
const pkcs11_attrib_model pkcs11_key_public_attributes[ ]
```

CKO_PUBLIC_KEY - Public Key Object Model

8.12.5.12 pkcs11_key_public_attributes_count

```
const CK_ULONG pkcs11_key_public_attributes_count = sizeof( pkcs11_key_public_attributes ) /
sizeof( pkcs11_key_public_attributes [0])
```

8.12.5.13 pkcs11_key_rsa_private_attributes

```
const pkcs11_attrib_model pkcs11_key_rsa_private_attributes[]
```

Initial value:

```
= {
    {  0x00000120UL ,            0
                },
    {  0x00000122UL ,    0
        },
    {  0x00000123UL ,   0
       },
    {  0x00000124UL ,            0
                },
    {  0x00000125UL ,            0
                },
    {  0x00000126UL ,         0
             },
    {  0x00000127UL ,         0
             },
    {  0x00000128UL ,        0
            },
}
```

CKO_PRIVATE_KEY (Type: CKK_RSA) - RSA Private Key Object Model

8.12.5.14 pkcs11_key_secret_attributes

```
const pkcs11_attrib_model pkcs11_key_secret_attributes[]
```

CKO_SECRET_KEY - Secret Key Object Base Model

8.12.5.15 pkcs11_key_secret_attributes_count

```
const CK_ULONG pkcs11_key_secret_attributes_count = sizeof( pkcs11_key_secret_attributes ) /
sizeof( pkcs11_key_secret_attributes [0])
```

8.12.5.16 pkcs11_lib_description

```
const char pkcs11_lib_description[] = "Cryptoauthlib PKCS11 Interface"
```

8.12.5.17 pkcs11_lib_manufacturer_id

```
const char pkcs11_lib_manufacturer_id[] = "Microchip Technology Inc"
```

8.12.5.18 pkcs11_object_cache

```
pkcs11_object_cache_t pkcs11_object_cache[PKCS11_MAX_OBJECTS_ALLOWED]
```

8.12.5.19 pkcs11_object_monotonic_attributes

```
const pkcs11_attrib_model pkcs11_object_monotonic_attributes[]
```

Initial value:

```
= {
    {  0x00000000UL ,           pkcs11_object_get_class
                                },
    {  0x00000300UL , pkcs11_object_get_type
                      },
    {  0x00000301UL ,   pkcs11_attrib_false
                        },
    {  0x00000302UL ,       pkcs11_attrib_false
                            },
    {  0x00000011UL ,            0
                               },
}
```

CKA_CLASS == CKO_HW_FEATURE_TYPE CKA_HW_FEATURE_TYPE == CKH_MONOTONIC_COUNTER

8.12.5.20 pkcs11_object_monotonic_attributes_count

```
const CK_ULONG pkcs11_object_monotonic_attributes_count = sizeof( pkcs11_object_monotonic_attributes
) / sizeof( pkcs11_object_monotonic_attributes [0])
```

8.13 TNG API (tng_)

These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.

8.13.0.1 TNG Functions

This folder has a number of convenience functions for working with TNG devices (currently ATECC608A-MAHTN-T).

These devices have standard certificates that can be easily read using the functions in tng_atcacert_client.h

Functions

- const atcacert_def_t ∗ tng_map_get_device_cert_def (int index)

    Helper function to iterate through all trust cert definitions.
- ATCA_STATUS tng_get_device_cert_def (const atcacert_def_t ∗∗cert_def)

    Get the TNG device certificate definition.
- ATCA_STATUS tng_get_device_pubkey (uint8_t ∗public_key)

    Uses GenKey command to calculate the public key from the primary device public key.

- const atcacert_def_t g_tflxtls_cert_def_4_device

- int tng_atcacert_max_device_cert_size (size_t ∗max_cert_size)

    Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t ∗cert, size_t ∗cert_size, const uint8_t ∗signer_cert)

    Reads the device certificate for a TNG device.
- int tng_atcacert_device_public_key (uint8_t ∗public_key, uint8_t ∗cert)

    Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t ∗max_cert_size)

    Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t ∗cert, size_t ∗cert_size)

    Reads the signer certificate for a TNG device.
- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

    Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

    Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

    Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

    Gets the root public key.

- const uint8_t g_cryptoauth_root_ca_002_cert [ ]
- const size_t g_cryptoauth_root_ca_002_cert_size
- #define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266

- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_1_signer

- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_2_device

- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_4_device
- #define TNGLORA_CERT_TEMPLATE_4_DEVICE_SIZE 552

- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_1_signer
- #define TNGTLS_CERT_TEMPLATE_1_SIGNER_SIZE 520

- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_2_device
- #define TNGTLS_CERT_TEMPLATE_2_DEVICE_SIZE 505
- #define TNGTLS_CERT_ELEMENTS_2_DEVICE_COUNT 2

- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_3_device
- #define TNGTLS_CERT_TEMPLATE_3_DEVICE_SIZE 546

8.13.1 Detailed Description

These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.

8.13.2 Macro Definition Documentation

8.13.2.1 CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET

```
#define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266
```

8.13.2.2 TNGLORA_CERT_TEMPLATE_4_DEVICE_SIZE

```
#define TNGLORA_CERT_TEMPLATE_4_DEVICE_SIZE 552
```

8.13.2.3 TNGTLS_CERT_ELEMENTS_2_DEVICE_COUNT

```
#define TNGTLS_CERT_ELEMENTS_2_DEVICE_COUNT 2
```

8.13.2.4 TNGTLS_CERT_TEMPLATE_1_SIGNER_SIZE

```
#define TNGTLS_CERT_TEMPLATE_1_SIGNER_SIZE 520
```

8.13.2.5 TNGTLS_CERT_TEMPLATE_2_DEVICE_SIZE

```
#define TNGTLS_CERT_TEMPLATE_2_DEVICE_SIZE 505
```

8.13.2.6 TNGTLS_CERT_TEMPLATE_3_DEVICE_SIZE

```
#define TNGTLS_CERT_TEMPLATE_3_DEVICE_SIZE 546
```

8.13.3 Function Documentation

8.13.3.1 tng_atcacert_device_public_key()

```
int tng_atcacert_device_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the device public key is used from this certificate. If set to NULL, the device public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.2 tng_atcacert_max_device_cert_size()

```
int tng_atcacert_max_device_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.3 tng_atcacert_max_signer_cert_size()

```
int tng_atcacert_max_signer_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.4 tng_atcacert_read_device_cert()

```
int tng_atcacert_read_device_cert (
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t * signer_cert )
```

Reads the device certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.
in	*signer_cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.5 tng_atcacert_read_signer_cert()

```
int tng_atcacert_read_signer_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Reads the signer certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.6 tng_atcacert_root_cert()

```
int tng_atcacert_root_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Get the TNG root cert.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.7 tng_atcacert_root_cert_size()

```
int tng_atcacert_root_cert_size (
            size_t * cert_size )
```

Get the size of the TNG root cert.

Parameters

out	*cert_size*	Certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.8 tng_atcacert_root_public_key()

```
int tng_atcacert_root_public_key (
            uint8_t * public_key )
```

Gets the root public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.9 tng_atcacert_signer_public_key()

```
int tng_atcacert_signer_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the signer public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

8.13.3.10 tng_get_device_cert_def()

```
ATCA_STATUS tng_get_device_cert_def (
            const atcacert_def_t ** cert_def )
```

Get the TNG device certificate definition.

Parameters

out	*cert_def*	TNG device certificate defnition is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.13.3.11 tng_get_device_pubkey()

```
ATCA_STATUS tng_get_device_pubkey (
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from the primary device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

8.13.3.12 tng_map_get_device_cert_def()

```
const atcacert_def_t* tng_map_get_device_cert_def (
            int index )
```

Helper function to iterate through all trust cert definitions.

Parameters

in	*index*	Map index

Returns

non-null value if success, otherwise NULL

8.13.4 Variable Documentation

8.13.4.1 g_cryptoauth_root_ca_002_cert

```
const uint8_t g_cryptoauth_root_ca_002_cert[]  [extern]
```

8.13.4.2 g_cryptoauth_root_ca_002_cert_size

```
const size_t g_cryptoauth_root_ca_002_cert_size  [extern]
```

8.13.4.3 g_tflxtls_cert_def_4_device

```
const atcacert_def_t g_tflxtls_cert_def_4_device  [extern]
```

8.13.4.4 g_tnglora_cert_def_1_signer

```
ATCA_DLL const atcacert_def_t g_tnglora_cert_def_1_signer
```

8.13.4.5 g_tnglora_cert_def_2_device

```
ATCA_DLL const atcacert_def_t g_tnglora_cert_def_2_device
```

8.13.4.6 g_tnglora_cert_def_4_device

```
ATCA_DLL const atcacert_def_t g_tnglora_cert_def_4_device
```

8.13.4.7 g_tngtls_cert_def_1_signer

```
ATCA_DLL const atcacert_def_t g_tngtls_cert_def_1_signer
```

8.13.4.8 g_tngtls_cert_def_2_device

```
ATCA_DLL const atcacert_def_t g_tngtls_cert_def_2_device
```

8.13.4.9 g_tngtls_cert_def_3_device

```
ATCA_DLL const atcacert_def_t g_tngtls_cert_def_3_device
```

Chapter 9

Data Structure Documentation

9.1 _ascii_kit_host_context Struct Reference

`#include <ascii_kit_host.h>`

Data Fields

- const atca_hal_kit_phy_t * phy
- uint8_t buffer [(2500)]
- ATCADevice device
- ATCAIfaceCfg ** iface
- size_t iface_count
- uint32_t flags

9.1.1 Field Documentation

9.1.1.1 buffer

```
uint8_t buffer[(2500)]
```

9.1.1.2 device

```
ATCADevice device
```

9.1.1.3 flags

```
uint32_t flags
```

9.1.1.4 iface

```
ATCAIfaceCfg** iface
```

9.1.1.5 iface_count

```
size_t iface_count
```

9.1.1.6 phy

```
const atca_hal_kit_phy_t* phy
```

9.2 _atecc508a_config Struct Reference

```
#include <atca_device.h>
```

Data Fields

- uint32_t SN03
- uint32_t RevNum
- uint32_t SN47
- uint8_t SN8
- uint8_t Reserved0
- uint8_t I2C_Enable
- uint8_t Reserved1
- uint8_t I2C_Address
- uint8_t Reserved2
- uint8_t OTPmode
- uint8_t ChipMode
- uint16_t SlotConfig [16]
- uint8_t Counter0 [8]
- uint8_t Counter1 [8]
- uint8_t LastKeyUse [16]
- uint8_t UserExtra
- uint8_t Selector
- uint8_t LockValue
- uint8_t LockConfig
- uint16_t SlotLocked
- uint16_t RFU
- uint32_t X509format
- uint16_t KeyConfig [16]

9.2.1 Field Documentation

9.2.1.1 ChipMode

```
uint8_t ChipMode
```

9.2.1.2 Counter0

```
uint8_t Counter0[8]
```

9.2.1.3 Counter1

```
uint8_t Counter1[8]
```

9.2.1.4 I2C_Address

```
uint8_t I2C_Address
```

9.2.1.5 I2C_Enable

```
uint8_t I2C_Enable
```

9.2.1.6 KeyConfig

```
uint16_t KeyConfig[16]
```

9.2.1.7 LastKeyUse

```
uint8_t LastKeyUse[16]
```

9.2.1.8 LockConfig

```
uint8_t LockConfig
```

9.2.1.9 LockValue

```
uint8_t LockValue
```

9.2.1.10 OTPmode

```
uint8_t OTPmode
```

9.2.1.11 Reserved0

```
uint8_t Reserved0
```

9.2.1.12 Reserved1

```
uint8_t Reserved1
```

9.2.1.13 Reserved2

```
uint8_t Reserved2
```

9.2.1.14 RevNum

```
uint32_t RevNum
```

9.2.1.15 RFU

```
uint16_t RFU
```

9.2.1.16 Selector

```
uint8_t Selector
```

9.2.1.17 SlotConfig

```
uint16_t SlotConfig[16]
```

9.2.1.18 SlotLocked

```
uint16_t SlotLocked
```

9.2.1.19 SN03

```
uint32_t SN03
```

9.2.1.20 SN47

```
uint32_t SN47
```

9.2.1.21 SN8

```
uint8_t SN8
```

9.2.1.22 UserExtra

```
uint8_t UserExtra
```

9.2.1.23 X509format

```
uint32_t X509format
```

9.3 _atecc608_config Struct Reference

```
#include <atca_device.h>
```

Data Fields

- uint32_t SN03
- uint32_t RevNum
- uint32_t SN47
- uint8_t SN8
- uint8_t AES_Enable
- uint8_t I2C_Enable
- uint8_t Reserved1
- uint8_t I2C_Address
- uint8_t Reserved2
- uint8_t CountMatch
- uint8_t ChipMode
- uint16_t SlotConfig [16]
- uint8_t Counter0 [8]
- uint8_t Counter1 [8]
- uint8_t UseLock
- uint8_t VolatileKeyPermission
- uint16_t SecureBoot
- uint8_t KdflvLoc
- uint16_t KdflvStr
- uint8_t Reserved3 [9]
- uint8_t UserExtra
- uint8_t UserExtraAdd
- uint8_t LockValue
- uint8_t LockConfig
- uint16_t SlotLocked
- uint16_t ChipOptions
- uint32_t X509format
- uint16_t KeyConfig [16]

9.3.1 Field Documentation

9.3.1.1 AES_Enable

```
uint8_t AES_Enable
```

9.3.1.2 ChipMode

```
uint8_t ChipMode
```

9.3.1.3 ChipOptions

```
uint16_t ChipOptions
```

9.3.1.4 Counter0

```
uint8_t Counter0[8]
```

9.3.1.5 Counter1

```
uint8_t Counter1[8]
```

9.3.1.6 CountMatch

```
uint8_t CountMatch
```

9.3.1.7 I2C_Address

```
uint8_t I2C_Address
```

9.3.1.8 I2C_Enable

```
uint8_t I2C_Enable
```

9.3.1.9 KdflvLoc

```
uint8_t KdflvLoc
```

9.3.1.10 KdflvStr

```
uint16_t KdflvStr
```

9.3.1.11 KeyConfig

```
uint16_t KeyConfig[16]
```

9.3.1.12 LockConfig

```
uint8_t LockConfig
```

9.3.1.13 LockValue

```
uint8_t LockValue
```

9.3.1.14 Reserved1

```
uint8_t Reserved1
```

9.3.1.15 Reserved2

```
uint8_t Reserved2
```

9.3.1.16 Reserved3

```
uint8_t Reserved3[9]
```

9.3.1.17 RevNum

```
uint32_t RevNum
```

9.3.1.18 SecureBoot

```
uint16_t SecureBoot
```

9.3.1.19 SlotConfig

```
uint16_t SlotConfig[16]
```

9.3.1.20 SlotLocked

```
uint16_t SlotLocked
```

9.3.1.21 SN03

```
uint32_t SN03
```

9.3.1.22 SN47

```
uint32_t SN47
```

9.3.1.23 SN8

```
uint8_t SN8
```

9.3.1.24 UseLock

```
uint8_t UseLock
```

9.3.1.25 UserExtra

```
uint8_t UserExtra
```

9.3.1.26 UserExtraAdd

```
uint8_t UserExtraAdd
```

9.3.1.27 VolatileKeyPermission

```
uint8_t VolatileKeyPermission
```

9.3.1.28 X509format

```
uint32_t X509format
```

9.4 _atsha204a_config Struct Reference

```
#include <atca_device.h>
```

Data Fields

- uint32_t SN03
- uint32_t RevNum
- uint32_t SN47
- uint8_t SN8
- uint8_t Reserved0
- uint8_t I2C_Enable
- uint8_t Reserved1
- uint8_t I2C_Address
- uint8_t Reserved2
- uint8_t OTPmode
- uint8_t ChipMode
- uint16_t SlotConfig [16]
- uint16_t Counter [8]
- uint8_t LastKeyUse [16]
- uint8_t UserExtra
- uint8_t Selector
- uint8_t LockValue
- uint8_t LockConfig

9.4.1 Field Documentation

9.4.1.1 ChipMode

```
uint8_t ChipMode
```

9.4.1.2 Counter

```
uint16_t Counter[8]
```

9.4.1.3 I2C_Address

```
uint8_t I2C_Address
```

9.4.1.4 I2C_Enable

```
uint8_t I2C_Enable
```

9.4.1.5 LastKeyUse

```
uint8_t LastKeyUse[16]
```

9.4.1.6 LockConfig

```
uint8_t LockConfig
```

9.4.1.7 LockValue

```
uint8_t LockValue
```

9.4.1.8 OTPmode

```
uint8_t OTPmode
```

9.4.1.9 Reserved0

```
uint8_t Reserved0
```

9.4.1.10 Reserved1

```
uint8_t Reserved1
```

9.4.1.11 Reserved2

```
uint8_t Reserved2
```

9.4.1.12 RevNum

```
uint32_t RevNum
```

9.4.1.13 Selector

```
uint8_t Selector
```

9.4.1.14 SlotConfig

```
uint16_t SlotConfig[16]
```

9.4.1.15 SN03

```
uint32_t SN03
```

9.4.1.16 SN47

```
uint32_t SN47
```

9.4.1.17 SN8

```
uint8_t SN8
```

9.4.1.18 UserExtra

```
uint8_t UserExtra
```

9.5 _kit_host_map_entry Struct Reference

```
#include <ascii_kit_host.h>
```

Data Fields

- const char * id
- ATCA_STATUS(* fp_command)(ascii_kit_host_context_t *ctx, int argc, char *argv[ ], uint8_t *response, size_t *rlen)

9.5.1 Detailed Description

Used to create command tables for the kit host parser

9.5.2 Field Documentation

9.5.2.1 fp_command

```
ATCA_STATUS(* fp_command) (ascii_kit_host_context_t *ctx, int argc, char *argv[ ], uint8_↩
t *response, size_t *rlen)
```

9.5.2.2 id

```
const char* id
```

9.6 _pcks11_mech_table_e Struct Reference

Data Fields

- CK_MECHANISM_TYPE type
- CK_MECHANISM_INFO info

9.6.1 Field Documentation

9.6.1.1 info

```
CK_MECHANISM_INFO info
```

9.6.1.2 type

```
CK_MECHANISM_TYPE type
```

9.7 _pkcs11_attrib_model Struct Reference

```
#include <pkcs11_attrib.h>
```

Data Fields

- const CK_ATTRIBUTE_TYPE type
- const attrib_f func

9.7.1 Field Documentation

9.7.1.1 func

```
const attrib_f func
```

9.7.1.2 type

```
const CK_ATTRIBUTE_TYPE type
```

9.8 _pkcs11_lib_ctx Struct Reference

```
#include <pkcs11_init.h>
```

Data Fields

- CK_BBOOL initialized
- CK_CREATEMUTEX create_mutex
- CK_DESTROYMUTEX destroy_mutex
- CK_LOCKMUTEX lock_mutex
- CK_UNLOCKMUTEX unlock_mutex
- CK_VOID_PTR mutex
- CK_VOID_PTR slots
- CK_ULONG slot_cnt
- CK_CHAR config_path [200]

9.8.1 Detailed Description

Library Context

9.8.2 Field Documentation

9.8.2.1 config_path

```
CK_CHAR config_path[200]
```

9.8.2.2 create_mutex

```
CK_CREATEMUTEX create_mutex
```

9.8.2.3 destroy_mutex

```
CK_DESTROYMUTEX destroy_mutex
```

9.8.2.4 initialized

```
CK_BBOOL initialized
```

9.8.2.5 lock_mutex

```
CK_LOCKMUTEX lock_mutex
```

9.8.2.6 mutex

```
CK_VOID_PTR mutex
```

9.8.2.7 slot_cnt

```
CK_ULONG slot_cnt
```

9.8.2.8 slots

```
CK_VOID_PTR slots
```

9.8.2.9 unlock_mutex

```
CK_UNLOCKMUTEX unlock_mutex
```

9.9 _pkcs11_object Struct Reference

```
#include <pkcs11_object.h>
```

Data Fields

- CK_OBJECT_CLASS class_id
- CK_ULONG class_type
- pkcs11_attrib_model const * attributes
- CK_ULONG count
- CK_ULONG size
- uint16_t slot
- CK_FLAGS flags
- CK_UTF8CHAR name [PKCS11_MAX_LABEL_SIZE+1]
- CK_VOID_PTR config
- CK_VOID_PTR data
- ta_element_attributes_t handle_info

9.9.1 Field Documentation

9.9.1.1 attributes

`pkcs11_attrib_model const* attributes`

List of attribute models this object possesses

9.9.1.2 class_id

`CK_OBJECT_CLASS class_id`

The Class Identifier

9.9.1.3 class_type

`CK_ULONG class_type`

The Class Type

9.9.1.4 config

`CK_VOID_PTR config`

9.9.1.5 count

`CK_ULONG count`

Count of attribute models

9.9.1.6 data

`CK_VOID_PTR data`

9.9.1.7 flags

`CK_FLAGS flags`

9.9.1.8 handle_info

```
ta_element_attributes_t handle_info
```

9.9.1.9 name

```
CK_UTF8CHAR name[PKCS11_MAX_LABEL_SIZE+1]
```

9.9.1.10 size

```
CK_ULONG size
```

9.9.1.11 slot

```
uint16_t slot
```

9.10 _pkcs11_object_cache_t Struct Reference

```
#include <pkcs11_object.h>
```

Data Fields

- CK_OBJECT_HANDLE handle
- pkcs11_object_ptr object

9.10.1 Field Documentation

9.10.1.1 handle

```
CK_OBJECT_HANDLE handle
```

Arbitrary (but unique) non-null identifier for an object

9.10.1.2 object

```
pkcs11_object_ptr object
```

The actual object

9.11 _pkcs11_session_ctx Struct Reference

```
#include <pkcs11_session.h>
```

Data Fields

- CK_BBOOL initialized
- pkcs11_slot_ctx_ptr slot
- CK_SESSION_HANDLE handle
- CK_STATE state
- CK_ULONG error
- CK_ATTRIBUTE_PTR attrib_list
- CK_ULONG attrib_count
- CK_ULONG object_index
- CK_ULONG object_count
- CK_OBJECT_HANDLE active_object
- CK_MECHANISM_TYPE active_mech
- pkcs11_session_mech_ctx active_mech_data

9.11.1 Detailed Description

Session Context

9.11.2 Field Documentation

9.11.2.1 active_mech

```
CK_MECHANISM_TYPE active_mech
```

9.11.2.2 active_mech_data

```
pkcs11_session_mech_ctx active_mech_data
```

9.11.2.3 active_object

```
CK_OBJECT_HANDLE active_object
```

9.11.2.4 attrib_count

```
CK_ULONG attrib_count
```

9.11.2.5 attrib_list

```
CK_ATTRIBUTE_PTR attrib_list
```

9.11.2.6 error

```
CK_ULONG error
```

9.11.2.7 handle

```
CK_SESSION_HANDLE handle
```

9.11.2.8 initialized

```
CK_BBOOL initialized
```

9.11.2.9 object_count

```
CK_ULONG object_count
```

9.11.2.10 object_index

```
CK_ULONG object_index
```

9.11.2.11 slot

```
pkcs11_slot_ctx_ptr slot
```

9.11.2.12 state

```
CK_STATE state
```

9.12 _pkcs11_session_mech_ctx Union Reference

```
#include <pkcs11_session.h>
```

Data Fields

- struct {
  atca_hmac_sha256_ctx_t context
  } hmac
- struct {
  atca_aes_cmac_ctx_t context
  } cmac
- struct {
  atca_aes_gcm_ctx_t context
  CK_BYTE tag_len
  } gcm

9.12.1 Field Documentation

9.12.1.1 cmac

```
struct { ... } cmac
```

9.12.1.2 context [1/3]

```
atca_hmac_sha256_ctx_t context
```

9.12.1.3 context [2/3]

```
atca_aes_cmac_ctx_t context
```

9.12.1.4 context [3/3]

```
atca_aes_gcm_ctx_t context
```

9.12.1.5 gcm

```
struct { ... } gcm
```

9.12.1.6 hmac

```
struct { ... } hmac
```

9.12.1.7 tag_len

```
CK_BYTE tag_len
```

9.13 _pkcs11_slot_ctx Struct Reference

```
#include <pkcs11_slot.h>
```

Data Fields

- CK_BBOOL initialized
- CK_SLOT_ID slot_id
- ATCADevice device_ctx
- ATCAIfaceCfg interface_config
- CK_SESSION_HANDLE session
- atecc608_config_t cfg_zone
- CK_FLAGS flags
- uint16_t user_pin_handle
- uint16_t so_pin_handle
- CK_UTF8CHAR label [PKCS11_MAX_LABEL_SIZE+1]
- CK_BBOOL logged_in
- CK_BYTE read_key [32]

9.13.1 Detailed Description

Slot Context

9.13.2 Field Documentation

9.13.2.1 cfg_zone

```
atecc608_config_t cfg_zone
```

9.13.2.2 device_ctx

```
ATCADevice device_ctx
```

9.13.2.3 flags

```
CK_FLAGS flags
```

9.13.2.4 initialized

```
CK_BBOOL initialized
```

9.13.2.5 interface_config

```
ATCAIfaceCfg interface_config
```

9.13.2.6 label

```
CK_UTF8CHAR label[PKCS11_MAX_LABEL_SIZE+1]
```

9.13.2.7 logged_in

```
CK_BBOOL logged_in
```

9.13.2.8 read_key

```
CK_BYTE read_key[32]
```

Accepted through C_Login as the user pin

9.13.2.9 session

```
CK_SESSION_HANDLE session
```

9.13.2.10 slot_id

```
CK_SLOT_ID slot_id
```

9.13.2.11 so_pin_handle

```
uint16_t so_pin_handle
```

9.13.2.12 user_pin_handle

```
uint16_t user_pin_handle
```

9.14 atca_aes_cbc_ctx Struct Reference

```
#include <atca_crypto_hw_aes.h>
```

Data Fields

- ATCADevice device

  Device Context Pointer.
- uint16_t key_id

  Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block

  Index of the 16-byte block to use within the key location for the actual key.
- uint8_t ciphertext [ATCA_AES128_BLOCK_SIZE]

  Ciphertext from last operation.

9.14.1 Field Documentation

9.14.1.1 ciphertext

`uint8_t ciphertext[ATCA_AES128_BLOCK_SIZE]`

Ciphertext from last operation.

9.14.1.2 device

`ATCADevice device`

Device Context Pointer.

9.14.1.3 key_block

`uint8_t key_block`

Index of the 16-byte block to use within the key location for the actual key.

9.14.1.4 key_id

`uint16_t key_id`

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

9.15 atca_aes_cbcmac_ctx Struct Reference

`#include <atca_crypto_hw_aes.h>`

Data Fields

- atca_aes_cbc_ctx_t cbc_ctx

  CBC context.
- uint8_t block_size

  Number of bytes in unprocessed block.
- uint8_t block [ATCA_AES128_BLOCK_SIZE]

  Unprocessed message storage.

9.15.1 Field Documentation

9.15.1.1 block

```
uint8_t block[ATCA_AES128_BLOCK_SIZE]
```

Unprocessed message storage.

9.15.1.2 block_size

```
uint8_t block_size
```

Number of bytes in unprocessed block.

9.15.1.3 cbc_ctx

```
atca_aes_cbc_ctx_t cbc_ctx
```

CBC context.

9.16 atca_aes_ccm_ctx Struct Reference

```
#include <atca_crypto_hw_aes.h>
```

Data Fields

- atca_aes_cbcmac_ctx_t cbc_mac_ctx

  CBC_MAC context.
- atca_aes_ctr_ctx_t ctr_ctx

  CTR context.
- uint8_t iv_size

  iv size
- uint8_t M

  Tag size.
- uint8_t counter [ATCA_AES128_BLOCK_SIZE]

  Initial counter value.
- uint8_t partial_aad [ATCA_AES128_BLOCK_SIZE]

  Partial blocks of data waiting to be processed.
- size_t partial_aad_size

  Amount of data in the partial block buffer.
- size_t text_size

  Size of data to be processed.
- uint8_t enc_cb [ATCA_AES128_BLOCK_SIZE]

  Last encrypted counter block.
- uint32_t data_size

  Size of the data being encrypted/decrypted in bytes.
- uint8_t ciphertext_block [ATCA_AES128_BLOCK_SIZE]

  Last ciphertext block.

9.16.1 Field Documentation

9.16.1.1 cbc_mac_ctx

```
atca_aes_cbcmac_ctx_t cbc_mac_ctx
```

CBC_MAC context.

9.16.1.2 ciphertext_block

```
uint8_t ciphertext_block[ATCA_AES128_BLOCK_SIZE]
```

Last ciphertext block.

9.16.1.3 counter

```
uint8_t counter[ATCA_AES128_BLOCK_SIZE]
```

Initial counter value.

9.16.1.4 ctr_ctx

```
atca_aes_ctr_ctx_t ctr_ctx
```

CTR context.

9.16.1.5 data_size

```
uint32_t data_size
```

Size of the data being encrypted/decrypted in bytes.

9.16.1.6 enc_cb

```
uint8_t enc_cb[ATCA_AES128_BLOCK_SIZE]
```

Last encrypted counter block.

9.16.1.7 iv_size

```
uint8_t iv_size
```

iv size

9.16.1.8 M

```
uint8_t M
```

Tag size.

9.16.1.9 partial_aad

```
uint8_t partial_aad[ATCA_AES128_BLOCK_SIZE]
```

Partial blocks of data waiting to be processed.

9.16.1.10 partial_aad_size

```
size_t partial_aad_size
```

Amount of data in the partial block buffer.

9.16.1.11 text_size

```
size_t text_size
```

Size of data to be processed.

9.17 atca_aes_cmac_ctx Struct Reference

```
#include <atca_crypto_hw_aes.h>
```

Data Fields

- atca_aes_cbc_ctx_t cbc_ctx

    CBC context.
- uint32_t block_size

    Number of bytes in current block.
- uint8_t block [ATCA_AES128_BLOCK_SIZE]

    Unprocessed message storage.

9.17.1 Field Documentation

9.17.1.1 block

```
uint8_t block[ATCA_AES128_BLOCK_SIZE]
```

Unprocessed message storage.

9.17.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

9.17.1.3 cbc_ctx

```
atca_aes_cbc_ctx_t cbc_ctx
```

CBC context.

9.18 atca_aes_ctr_ctx Struct Reference

```
#include <atca_crypto_hw_aes.h>
```

Data Fields

- ATCADevice device
  Device Context Pointer.
- uint16_t key_id
  Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block
  Index of the 16-byte block to use within the key location for the actual key.
- uint8_t cb [ATCA_AES128_BLOCK_SIZE]
  Counter block, comprises of nonce + count value (16 bytes).
- uint8_t counter_size
  Size of counter in the initialization vector.

9.18.1 Field Documentation

9.18.1.1 cb

```
uint8_t cb[ATCA_AES128_BLOCK_SIZE]
```

Counter block, comprises of nonce + count value (16 bytes).

9.18.1.2 counter_size

```
uint8_t counter_size
```

Size of counter in the initialization vector.

9.18.1.3 device

```
ATCADevice device
```

Device Context Pointer.

9.18.1.4 key_block

```
uint8_t key_block
```

Index of the 16-byte block to use within the key location for the actual key.

9.18.1.5 key_id

```
uint16_t key_id
```

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

9.19 atca_aes_gcm_ctx Struct Reference

```
#include <calib_aes_gcm.h>
```

Data Fields

- uint16_t key_id
  Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block
  Index of the 16-byte block to use within the key location for the actual key.
- uint8_t cb [AES_DATA_SIZE]
  Counter block, comprises of nonce + count value (16 bytes).
- uint32_t data_size
  Size of the data being encrypted/decrypted in bytes.
- uint32_t aad_size
  Size of the additional authenticated data in bytes.
- uint8_t h [AES_DATA_SIZE]
  Subkey for ghash functions in GCM.
- uint8_t j0 [AES_DATA_SIZE]
  Precounter block generated from IV.
- uint8_t y [AES_DATA_SIZE]
  Current GHASH output.
- uint8_t partial_aad [AES_DATA_SIZE]
  Partial blocks of data waiting to be processed.
- uint32_t partial_aad_size
  Amount of data in the partial block buffer.
- uint8_t enc_cb [AES_DATA_SIZE]
  Last encrypted counter block.
- uint8_t ciphertext_block [AES_DATA_SIZE]
  Last ciphertext block.

9.19.1 Detailed Description

Context structure for AES GCM operations.

9.19.2 Field Documentation

9.19.2.1 aad_size

```
uint32_t aad_size
```

Size of the additional authenticated data in bytes.

9.19.2.2 cb

```
uint8_t cb[AES_DATA_SIZE]
```

Counter block, comprises of nonce + count value (16 bytes).

9.19.2.3 ciphertext_block

```
uint8_t ciphertext_block[AES_DATA_SIZE]
```

Last ciphertext block.

9.19.2.4 data_size

```
uint32_t data_size
```

Size of the data being encrypted/decrypted in bytes.

9.19.2.5 enc_cb

```
uint8_t enc_cb[AES_DATA_SIZE]
```

Last encrypted counter block.

9.19.2.6 h

```
uint8_t h[AES_DATA_SIZE]
```

Subkey for ghash functions in GCM.

9.19.2.7 j0

```
uint8_t j0[AES_DATA_SIZE]
```

Precounter block generated from IV.

9.19.2.8 key_block

```
uint8_t key_block
```

Index of the 16-byte block to use within the key location for the actual key.

9.19.2.9 key_id

```
uint16_t key_id
```

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

9.19.2.10 partial_aad

```
uint8_t partial_aad[AES_DATA_SIZE]
```

Partial blocks of data waiting to be processed.

9.19.2.11 partial_aad_size

```
uint32_t partial_aad_size
```

Amount of data in the partial block buffer.

9.19.2.12 y

```
uint8_t y[AES_DATA_SIZE]
```

Current GHASH output.

9.20 atca_check_mac_in_out Struct Reference

Input/output parameters for function atcah_check_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] CheckMac command Mode
- uint16_t key_id

  [in] CheckMac command KeyID
- const uint8_t ∗ sn

  [in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ client_chal

  [in] ClientChal data, 32 bytes. Can be NULL if mode[0] is 1.
- uint8_t ∗ client_resp

  [out] Calculated ClientResp will be returned here.
- const uint8_t ∗ other_data

  [in] OtherData, 13 bytes
- const uint8_t ∗ otp

  [in] First 8 bytes of the OTP zone data. Can be NULL is mode[5] is 0.
- const uint8_t ∗ slot_key
- const uint8_t ∗ target_key
- struct atca_temp_key ∗ temp_key

  [in,out] Current state of TempKey. Required if mode[0] or mode[1] are 1.

9.20.1 Detailed Description

Input/output parameters for function atcah_check_mac().

9.20.2 Field Documentation

9.20.2.1 client_chal

```
const uint8_t* client_chal
```

[in] ClientChal data, 32 bytes. Can be NULL if mode[0] is 1.

9.20.2.2 client_resp

```
uint8_t* client_resp
```

[out] Calculated ClientResp will be returned here.

9.20.2.3 key_id

```
uint16_t key_id
```

[in] CheckMac command KeyID

9.20.2.4 mode

```
uint8_t mode
```

[in] CheckMac command Mode

9.20.2.5 other_data

```
const uint8_t* other_data
```

[in] OtherData, 13 bytes

9.20.2.6 otp

```
const uint8_t* otp
```

[in] First 8 bytes of the OTP zone data. Can be NULL is mode[5] is 0.

9.20.2.7 slot_key

```
const uint8_t* slot_key
```

[in] 32 byte key value in the slot specified by slot_id. Can be NULL if mode[1] is 1.

9.20.2.8 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

9.20.2.9 target_key

```
const uint8_t* target_key
```

[in] If this is not NULL, it assumes CheckMac copy is enabled for the specified key_id (ReadKey=0). If key_id is even, this should be the 32-byte key value for the slot key_id+1, otherwise this should be set to slot_key.

9.20.2.10 temp_key

```
struct atca_temp_key* temp_key
```

[in,out] Current state of TempKey. Required if mode[0] or mode[1] are 1.

9.21 atca_decrypt_in_out Struct Reference

Input/output parameters for function atca_decrypt().

```
#include <atca_host.h>
```

Data Fields

- uint8_t * crypto_data

    [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct atca_temp_key * temp_key

    [in,out] Pointer to TempKey structure.

9.21.1 Detailed Description

Input/output parameters for function atca_decrypt().

9.22 atca_derive_key_in_out Struct Reference

Input/output parameters for function atcah_derive_key().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode
  Mode (param 1) of the derive key command.
- uint16_t target_key_id
  Key ID (param 2) of the target slot to run the command on.
- const uint8_t * sn
  Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t * parent_key
  Parent key to be used in the derive key calculation (32 bytes).
- uint8_t * target_key
  Derived key will be returned here (32 bytes).
- struct atca_temp_key * temp_key
  Current state of TempKey.

9.22.1 Detailed Description

Input/output parameters for function atcah_derive_key().

9.22.2 Field Documentation

9.22.2.1 mode

```
uint8_t mode
```

Mode (param 1) of the derive key command.

9.22.2.2 parent_key

```
const uint8_t* parent_key
```

Parent key to be used in the derive key calculation (32 bytes).

9.22.2.3 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

9.22.2.4 target_key

```
uint8_t* target_key
```

Derived key will be returned here (32 bytes).

9.22.2.5 target_key_id

```
uint16_t target_key_id
```

Key ID (param 2) of the target slot to run the command on.

9.22.2.6 temp_key

```
struct atca_temp_key* temp_key
```

Current state of TempKey.

9.23 atca_derive_key_mac_in_out Struct Reference

Input/output parameters for function atcah_derive_key_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode
  Mode (param 1) of the derive key command.
- uint16_t target_key_id
  Key ID (param 2) of the target slot to run the command on.
- const uint8_t ∗ sn
  Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ parent_key
  Parent key to be used in the derive key calculation (32 bytes).
- uint8_t ∗ mac
  DeriveKey MAC will be returned here.

9.23.1 Detailed Description

Input/output parameters for function atcah_derive_key_mac().

9.23.2 Field Documentation

9.23.2.1 mac

```
uint8_t* mac
```

DeriveKey MAC will be returned here.

9.23.2.2 mode

```
uint8_t mode
```

Mode (param 1) of the derive key command.

9.23.2.3 parent_key

```
const uint8_t* parent_key
```

Parent key to be used in the derive key calculation (32 bytes).

9.23.2.4 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

9.23.2.5 target_key_id

```
uint16_t target_key_id
```

Key ID (param 2) of the target slot to run the command on.

9.24 atca_device Struct Reference

atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods

```
#include <atca_device.h>
```

Data Fields

- atca_iface_t mIface
- uint8_t device_state
- uint8_t clock_divider
- uint16_t execution_time_msec
- uint8_t session_state
- uint16_t session_counter
- uint16_t session_key_id
- uint8_t * session_key
- uint8_t session_key_len
- uint16_t options

9.24.1 Detailed Description

atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods

9.24.2 Field Documentation

9.24.2.1 clock_divider

```
uint8_t clock_divider
```

9.24.2.2 device_state

```
uint8_t device_state
```

Device Power State

9.24.2.3 execution_time_msec

```
uint16_t execution_time_msec
```

9.24.2.4 mIface

```
atca_iface_t mIface
```

Physical interface

9.24.2.5 options

```
uint16_t options
```

Nested command details parameter

9.24.2.6 session_counter

```
uint16_t session_counter
```

Secure Session Message Count

9.24.2.7 session_key

```
uint8_t* session_key
```

Session Key

9.24.2.8 session_key_id

```
uint16_t session_key_id
```

Key ID used for a secure sesison

9.24.2.9 session_key_len

```
uint8_t session_key_len
```

Length of key used for the session in bytes

9.24.2.10 session_state

```
uint8_t session_state
```

Secure Session State

9.25 atca_gen_dig_in_out Struct Reference

Input/output parameters for function atcah_gen_dig().

```
#include <atca_host.h>
```

Data Fields

- uint8_t zone
  [in] Zone/Param1 for the GenDig command
- uint16_t key_id
  [in] KeyId/Param2 for the GenDig command
- uint16_t slot_conf
  [in] Slot config for the GenDig command
- uint16_t key_conf
  [in] Key config for the GenDig command
- uint8_t slot_locked
  [in] slot locked for the GenDig command
- uint32_t counter
  [in] counter for the GenDig command
- bool is_key_nomac
  [in] Set to true if the slot pointed to be key_id has the SotConfig.NoMac bit set
- const uint8_t ∗ sn
  [in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ stored_value
  [in] 32-byte slot value, config block, OTP block as specified by the Zone/KeyId parameters
- const uint8_t ∗ other_data
  [in] 32-byte value for shared nonce zone, 4-byte value if is_key_nomac is true, ignored and/or NULL otherwise
- struct atca_temp_key ∗ temp_key
  [inout] Current state of TempKey

9.25.1 Detailed Description

Input/output parameters for function atcah_gen_dig().

9.25.2 Field Documentation

9.25.2.1 counter

```
uint32_t counter
```

[in] counter for the GenDig command

9.25.2.2 is_key_nomac

```
bool is_key_nomac
```

[in] Set to true if the slot pointed to be key_id has the SotConfig.NoMac bit set

9.25.2.3 key_conf

```
uint16_t key_conf
```

[in] Key config for the GenDig command

9.25.2.4 key_id

```
uint16_t key_id
```

[in] KeyId/Param2 for the GenDig command

9.25.2.5 other_data

```
const uint8_t* other_data
```

[in] 32-byte value for shared nonce zone, 4-byte value if is_key_nomac is true, ignored and/or NULL otherwise

9.25.2.6 slot_conf

```
uint16_t slot_conf
```

[in] Slot config for the GenDig command

9.25.2.7 slot_locked

```
uint8_t slot_locked
```

[in] slot locked for the GenDig command

9.25.2.8 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

9.25.2.9 stored_value

```
const uint8_t* stored_value
```

[in] 32-byte slot value, config block, OTP block as specified by the Zone/KeyId parameters

9.25.2.10 temp_key

```
struct atca_temp_key* temp_key
```

[inout] Current state of TempKey

9.25.2.11 zone

```
uint8_t zone
```

[in] Zone/Param1 for the GenDig command

9.26 atca_gen_key_in_out Struct Reference

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    [in] GenKey Mode
- uint16_t key_id

    [in] GenKey KeyID
- const uint8_t * public_key

    [in] Public key to be used in the PubKey digest. X and Y integers in big-endian format. 64 bytes for P256 curve.
- size_t public_key_size

    [in] Total number of bytes in the public key. 64 bytes for P256 curve.
- const uint8_t * other_data

    [in] 3 bytes required when bit 4 of the mode is set. Can be NULL otherwise.
- const uint8_t * sn

    [in] Device serial number SN[0:8] (9 bytes). Only SN[0:1] and SN[8] are required though.
- struct atca_temp_key * temp_key

    [in,out] As input the current state of TempKey. As output, the resulting PubKEy digest.

9.26.1 Detailed Description

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

9.26.2 Field Documentation

9.26.2.1 key_id

```
uint16_t key_id
```

[in] GenKey KeyID

9.26.2.2 mode

```
uint8_t mode
```

[in] GenKey Mode

9.26.2.3 other_data

```
const uint8_t* other_data
```

[in] 3 bytes required when bit 4 of the mode is set. Can be NULL otherwise.

9.26.2.4 public_key

```
const uint8_t* public_key
```

[in] Public key to be used in the PubKey digest. X and Y integers in big-endian format. 64 bytes for P256 curve.

9.26.2.5 public_key_size

```
size_t public_key_size
```

[in] Total number of bytes in the public key. 64 bytes for P256 curve.

9.26.2.6 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8] (9 bytes). Only SN[0:1] and SN[8] are required though.

9.26.2.7 temp_key

```
struct atca_temp_key* temp_key
```

[in,out] As input the current state of TempKey. As output, the resulting PubKEy digest.

9.27 atca_hal_kit_phy_t Struct Reference

```
#include <atca_hal.h>
```

Data Fields

- ATCA_STATUS(∗ send)(void ∗ctx, uint8_t ∗txdata, uint16_t txlen)
- ATCA_STATUS(∗ recv)(void ∗ctx, uint8_t ∗rxdata, uint16_t ∗rxlen)
- void ∗(∗ packet_alloc)(size_t bytes)
- void(∗ packet_free)(void ∗packet)
- void ∗ hal_data

9.27.1 Field Documentation

9.27.1.1 hal_data

```
void* hal_data
```

Physical layer context

9.27.1.2 packet_alloc

```
void*(* packet_alloc) (size_t bytes)
```

Allocate a phy packet

9.27.1.3 packet_free

```
void(* packet_free) (void *packet)
```

Free a phy packet

9.27.1.4 recv

```
ATCA_STATUS(* recv) (void *ctx, uint8_t *rxdata, uint16_t *rxlen)
```

Must be a blocking receive

9.27.1.5 send

```
ATCA_STATUS(* send) (void *ctx, uint8_t *txdata, uint16_t txlen)
```

Must be a blocking send

9.28 atca_hal_list_entry_t Struct Reference

Structure that holds the hal/phy maping for different interface types.

Data Fields

- uint8_t iface_type
- ATCAHAL_t * hal
- ATCAHAL_t * phy

9.28.1 Detailed Description

Structure that holds the hal/phy maping for different interface types.

9.28.2 Field Documentation

9.28.2.1 hal

```
ATCAHAL_t* hal
```

9.28.2.2 iface_type

```
uint8_t iface_type
```

9.28.2.3 phy

```
ATCAHAL_t* phy
```

Physical interface for the specific HAL

9.29 atca_hmac_in_out Struct Reference

Input/output parameters for function atca_hmac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode
  [in] Mode parameter used in HMAC command (Param1).
- uint16_t key_id
  [in] KeyID parameter used in HMAC command (Param2).
- const uint8_t * key
  [in] Pointer to 32-byte key used to generate HMAC digest.
- const uint8_t * otp
  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.
- const uint8_t * sn
  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.
- uint8_t * response
  [out] Pointer to 32-byte SHA-256 HMAC digest.
- struct atca_temp_key * temp_key
  [in,out] Pointer to TempKey structure.

9.29.1 Detailed Description

Input/output parameters for function atca_hmac().

9.30 atca_i2c_host_s Struct Reference

Data Fields

- char i2c_file [16]
- int ref_ct

9.30.1 Field Documentation

9.30.1.1 i2c_file

```
char i2c_file[16]
```

9.30.1.2 ref_ct

```
int ref_ct
```

9.31 atca_iface Struct Reference

atca_iface is the context structure for a configured interface

```
#include <atca_iface.h>
```

Data Fields

- ATCAIfaceCfg * mIfaceCFG
- ATCAHAL_t * hal
- ATCAHAL_t * phy
- void * hal_data

9.31.1 Detailed Description

atca_iface is the context structure for a configured interface

9.31.2 Field Documentation

9.31.2.1 hal

```
ATCAHAL_t* hal
```

The configured HAL for the interface

9.31.2.2 hal_data

```
void* hal_data
```

Pointer to HAL specific context/data

9.31.2.3 mIfaceCFG

```
ATCAIfaceCfg* mIfaceCFG
```

Points to previous defined/given Cfg object, the caller manages this

9.31.2.4 phy

```
ATCAHAL_t* phy
```

When a HAL is not a "native" hal it needs a physical layer to be associated with it

9.32 atca_include_data_in_out Struct Reference

Input / output parameters for function atca_include_data().

```
#include <atca_host.h>
```

Data Fields

- uint8_t * p_temp

  [out] pointer to output buffer
- const uint8_t * otp

  [in] pointer to one-time-programming data
- const uint8_t * sn

  [in] pointer to serial number data
- uint8_t mode

9.32.1 Detailed Description

Input / output parameters for function atca_include_data().

9.32.2 Field Documentation

9.32.2.1 mode

```
uint8_t mode
```

9.33 atca_io_decrypt_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- const uint8_t * io_key

  IO protection key (32 bytes).
- const uint8_t * out_nonce

  OutNonce returned from command (32 bytes).
- uint8_t * data

  As input, encrypted data. As output, decrypted data.
- size_t data_size

  Size of data in bytes (32 or 64).

9.33.1 Field Documentation

9.33.1.1 data

```
uint8_t* data
```

As input, encrypted data. As output, decrypted data.

9.33.1.2 data_size

```
size_t data_size
```

Size of data in bytes (32 or 64).

9.33.1.3 io_key

```
const uint8_t* io_key
```

IO protection key (32 bytes).

9.33.1.4 out_nonce

```
const uint8_t* out_nonce
```

OutNonce returned from command (32 bytes).

9.34 atca_jwt_t Struct Reference

Structure to hold metadata information about the jwt being built.

```
#include <atca_jwt.h>
```

Data Fields

- char * buf
- uint16_t buflen
- uint16_t cur

9.34.1 Detailed Description

Structure to hold metadata information about the jwt being built.

9.34.2 Field Documentation

9.34.2.1 buf

```
char* buf
```

9.34.2.2 buflen

```
uint16_t buflen
```

9.34.2.3 cur

```
uint16_t cur
```

9.35 atca_mac_in_out Struct Reference

Input/output parameters for function atca_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    [in] Mode parameter used in MAC command (Param1).
- uint16_t key_id

    [in] KeyID parameter used in MAC command (Param2).
- const uint8_t ∗ challenge

    [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.
- const uint8_t ∗ key

    [in] Pointer to 32-byte key used to generate MAC digest.
- const uint8_t ∗ otp

    [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- const uint8_t ∗ sn

    [in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.
- uint8_t ∗ response

    [out] Pointer to 32-byte SHA-256 digest (MAC).
- struct atca_temp_key ∗ temp_key

    [in,out] Pointer to TempKey structure.

9.35.1 Detailed Description

Input/output parameters for function atca_mac().

9.36 atca_mbedtls_eckey_s Struct Reference

```
#include <atca_mbedtls_wrap.h>
```

Data Fields

- ATCADevice device
- uint16_t handle

9.36.1 Detailed Description

Structure to hold metadata - is written into the mbedtls pk structure as the private key bignum value 'd' which otherwise would be unused. Bignums can be any arbitrary length of bytes

9.36.2 Field Documentation

9.36.2.1 device

```
ATCADevice device
```

9.36.2.2 handle

```
uint16_t handle
```

9.37 atca_nonce_in_out Struct Reference

Input/output parameters for function atca_nonce().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] Mode parameter used in Nonce command (Param1).
- uint16_t zero

  [in] Zero parameter used in Nonce command (Param2).
- const uint8_t * num_in

  [in] Pointer to 20-byte NumIn data used in Nonce command.
- const uint8_t * rand_out

  [in] Pointer to 32-byte RandOut data from Nonce command.
- struct atca_temp_key * temp_key

  [in,out] Pointer to TempKey structure.

9.37.1 Detailed Description

Input/output parameters for function atca_nonce().

9.38 atca_secureboot_enc_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- const uint8_t * io_key

  IO protection key value (32 bytes)
- const struct atca_temp_key * temp_key

  Current value of TempKey.
- const uint8_t * digest

  Plaintext digest as input.
- uint8_t * hashed_key

  Calculated key is returned here (32 bytes)
- uint8_t * digest_enc

  Encrypted (ciphertext) digest is return here (32 bytes)

9.38.1 Field Documentation

9.38.1.1 digest

```
const uint8_t* digest
```

Plaintext digest as input.

9.38.1.2 digest_enc

```
uint8_t* digest_enc
```

Encrypted (ciphertext) digest is return here (32 bytes)

9.38.1.3 hashed_key

```
uint8_t* hashed_key
```

Calculated key is returned here (32 bytes)

9.38.1.4 io_key

```
const uint8_t* io_key
```

IO protection key value (32 bytes)

9.38.1.5 temp_key

```
const struct atca_temp_key* temp_key
```

Current value of TempKey.

9.39 atca_secureboot_mac_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    SecureBoot mode (param1)
- uint16_t param2

    SecureBoot param2.
- uint16_t secure_boot_config

    SecureBootConfig value from configuration zone.
- const uint8_t * hashed_key

    Hashed key. SHA256(IO Protection Key | TempKey)
- const uint8_t * digest

    Digest (unencrypted)
- const uint8_t * signature

    Signature (can be NULL if not required)
- uint8_t * mac

    MAC is returned here.

9.39.1 Field Documentation

9.39.1.1 digest

```
const uint8_t* digest
```

Digest (unencrypted)

9.39.1.2 hashed_key

```
const uint8_t* hashed_key
```

Hashed key. SHA256(IO Protection Key | TempKey)

9.39.1.3 mac

```
uint8_t* mac
```

MAC is returned here.

9.39.1.4 mode

```
uint8_t mode
```

SecureBoot mode (param1)

9.39.1.5 param2

```
uint16_t param2
```

SecureBoot param2.

9.39.1.6 secure_boot_config

```
uint16_t secure_boot_config
```

SecureBootConfig value from configuration zone.

9.39.1.7 signature

```
const uint8_t* signature
```

Signature (can be NULL if not required)

9.40 atca_session_key_in_out Struct Reference

Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

```
#include <atca_host.h>
```

Data Fields

- uint8_t * transport_key
- uint16_t transport_key_id
- const uint8_t * sn
- uint8_t * nonce
- uint8_t * session_key

9.40.1 Detailed Description

Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

9.40.2 Field Documentation

9.40.2.1 nonce

```
uint8_t* nonce
```

9.40.2.2 session_key

```
uint8_t* session_key
```

9.40.2.3 sn

```
const uint8_t* sn
```

9.40.2.4 transport_key

```
uint8_t* transport_key
```

9.40.2.5 transport_key_id

```
uint16_t transport_key_id
```

9.41 atca_sha256_ctx Struct Reference

```
#include <calib_basic.h>
```

Data Fields

- uint32_t total_msg_size

    Total number of message bytes processed.
- uint32_t block_size

    Number of bytes in current block.
- uint8_t block [ATCA_SHA256_BLOCK_SIZE *2]

    Unprocessed message storage.

9.41.1 Field Documentation

9.41.1.1 block

```
uint8_t block[ATCA_SHA256_BLOCK_SIZE *2]
```

Unprocessed message storage.

9.41.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

9.41.1.3 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

9.42 atca_sign_internal_in_out Struct Reference

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] Sign Mode
- uint16_t key_id

  [in] Sign KeyID
- uint16_t slot_config

  [in] SlotConfig[TempKeyFlags.keyId]
- uint16_t key_config

  [in] KeyConfig[TempKeyFlags.keyId]
- uint8_t use_flag

  [in] UseFlag[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A
- uint8_t update_count

  [in] UpdateCount[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A
- bool is_slot_locked

  [in] Is TempKeyFlags.keyId slot locked.
- bool for_invalidate

  [in] Set to true if this will be used for the Verify(Invalidate) command.
- const uint8_t * sn

  [in] Device serial number SN[0:8] (9 bytes)
- const struct atca_temp_key * temp_key

  [in] The current state of TempKey.
- uint8_t * message

  [out] Full 55 byte message the Sign(internal) command will build. Can be NULL if not required.
- uint8_t * verify_other_data

  [out] The 19 byte OtherData bytes to be used with the Verify(In/Validate) command. Can be NULL if not required.
- uint8_t * digest

  [out] SHA256 digest of the full 55 byte message. Can be NULL if not required.

9.42.1 Detailed Description

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

9.42.2 Field Documentation

9.42.2.1 digest

```
uint8_t* digest
```

[out] SHA256 digest of the full 55 byte message. Can be NULL if not required.

9.42.2.2 for_invalidate

```
bool for_invalidate
```

[in] Set to true if this will be used for the Verify(Invalidate) command.

9.42.2.3 is_slot_locked

```
bool is_slot_locked
```

[in] Is TempKeyFlags.keyId slot locked.

9.42.2.4 key_config

```
uint16_t key_config
```

[in] KeyConfig[TempKeyFlags.keyId]

9.42.2.5 key_id

```
uint16_t key_id
```

[in] Sign KeyID

9.42.2.6 message

```
uint8_t* message
```

[out] Full 55 byte message the Sign(internal) command will build. Can be NULL if not required.

9.42.2.7 mode

```
uint8_t mode
```

[in] Sign Mode

9.42.2.8 slot_config

```
uint16_t slot_config
```

[in] SlotConfig[TempKeyFlags.keyId]

9.42.2.9 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8] (9 bytes)

9.42.2.10 temp_key

```
const struct atca_temp_key* temp_key
```

[in] The current state of TempKey.

9.42.2.11 update_count

```
uint8_t update_count
```

[in] UpdateCount[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A

9.42.2.12 use_flag

```
uint8_t use_flag
```

[in] UseFlag[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A

9.42.2.13 verify_other_data

```
uint8_t* verify_other_data
```

[out] The 19 byte OtherData bytes to be used with the Verify(In/Validate) command. Can be NULL if not required.

9.43 atca_spi_host_s Struct Reference

Data Fields

- char spi_file [20]
- int f_spi

9.43.1 Field Documentation

9.43.1.1 f_spi

```
int f_spi
```

9.43.1.2 spi_file

```
char spi_file[20]
```

9.44 atca_temp_key Struct Reference

Structure to hold TempKey fields.

```
#include <atca_host.h>
```

Data Fields

- uint8_t value [ATCA_KEY_SIZE *2]

  Value of TempKey (64 bytes for ATECC608 only)
- unsigned key_id: 4

  If TempKey was derived from a slot or transport key (GenDig or GenKey), that key ID is saved here.
- unsigned source_flag: 1

  Indicates id TempKey started from a random nonce (0) or not (1).
- unsigned gen_dig_data: 1

  TempKey was derived from the GenDig command.
- unsigned gen_key_data: 1

  TempKey was derived from the GenKey command (ATECC devices only).
- unsigned no_mac_flag: 1

  TempKey was derived from a key that has the NoMac bit set preventing the use of the MAC command. Known as CheckFlag in ATSHA devices).
- unsigned valid: 1

  TempKey is valid.
- uint8_t is_64

  TempKey has 64 bytes of valid data.

9.44.1 Detailed Description

Structure to hold TempKey fields.

9.44.2 Field Documentation

9.44.2.1 gen_dig_data

```
unsigned gen_dig_data
```

TempKey was derived from the GenDig command.

9.44.2.2 gen_key_data

```
unsigned gen_key_data
```

TempKey was derived from the GenKey command (ATECC devices only).

9.44.2.3 is_64

```
uint8_t is_64
```

TempKey has 64 bytes of valid data.

9.44.2.4 key_id

```
unsigned key_id
```

If TempKey was derived from a slot or transport key (GenDig or GenKey), that key ID is saved here.

9.44.2.5 no_mac_flag

```
unsigned no_mac_flag
```

TempKey was derived from a key that has the NoMac bit set preventing the use of the MAC command. Known as CheckFlag in ATSHA devices).

9.44.2.6 source_flag

```
unsigned source_flag
```

Indicates id TempKey started from a random nonce (0) or not (1).

9.44.2.7 valid

```
unsigned valid
```

TempKey is valid.

9.44.2.8 value

```
uint8_t value[ATCA_KEY_SIZE *2]
```

Value of TempKey (64 bytes for ATECC608 only)

9.45 atca_uart_host_s Struct Reference

Data Fields

- char uart_file [20]
- int fd_uart
- int uart_baud
- int uart_wordsize
- uint8_t uart_parity
- uint8_t uart_stopbit
- int ref_ct
- HANDLE hSerial

9.45.1 Field Documentation

9.45.1.1 fd_uart

```
int fd_uart
```

9.45.1.2 hSerial

```
HANDLE hSerial
```

9.45.1.3 ref_ct

```
int ref_ct
```

9.45.1.4 uart_baud

```
int uart_baud
```

9.45.1.5 uart_file

```
char uart_file
```

9.45.1.6 uart_parity

```
uint8_t uart_parity
```

9.45.1.7 uart_stopbit

```
uint8_t uart_stopbit
```

9.45.1.8 uart_wordsize

```
int uart_wordsize
```

9.46 atca_verify_in_out Struct Reference

Input/output parameters for function atcah_verify().

```
#include <atca_host.h>
```

Data Fields

- uint16_t curve_type

    [in] Curve type used in Verify command (Param2).
- const uint8_t * signature

    [in] Pointer to ECDSA signature to be verified
- const uint8_t * public_key

    [in] Pointer to the public key to be used for verification
- struct atca_temp_key * temp_key

    [in,out] Pointer to TempKey structure.

9.46.1 Detailed Description

Input/output parameters for function atcah_verify().

9.47 atca_verify_mac Struct Reference

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    Mode (Param1) parameter used in Verify command.
- uint16_t key_id

    KeyID (Param2) used in Verify command.
- const uint8_t * signature

    Signature used in Verify command (64 bytes).
- const uint8_t * other_data

    OtherData used in Verify command (19 bytes).
- const uint8_t * msg_dig_buf

    Message digest buffer (64 bytes).
- const uint8_t * io_key

    IO protection key value (32 bytes).
- const uint8_t * sn

    Serial number (9 bytes).
- const atca_temp_key_t * temp_key

    TempKey.
- uint8_t * mac

    Calculated verification MAC is returned here (32 bytes).

9.47.1 Field Documentation

9.47.1.1 io_key

```
const uint8_t* io_key
```

IO protection key value (32 bytes).

9.47.1.2 key_id

```
uint16_t key_id
```

KeyID (Param2) used in Verify command.

9.47.1.3 mac

```
uint8_t* mac
```

Calculated verification MAC is returned here (32 bytes).

9.47.1.4 mode

```
uint8_t mode
```

Mode (Param1) parameter used in Verify command.

9.47.1.5 msg_dig_buf

```
const uint8_t* msg_dig_buf
```

Message digest buffer (64 bytes).

9.47.1.6 other_data

```
const uint8_t* other_data
```

OtherData used in Verify command (19 bytes).

9.47.1.7 signature

```
const uint8_t* signature
```

Signature used in Verify command (64 bytes).

9.47.1.8 sn

```
const uint8_t* sn
```

Serial number (9 bytes).

9.47.1.9 temp_key

```
const atca_temp_key_t* temp_key
```

TempKey.

9.48 atca_write_mac_in_out Struct Reference

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t zone

    Zone/Param1 for the Write or PrivWrite command.
- uint16_t key_id

    KeyID/Param2 for the Write or PrivWrite command.
- const uint8_t * sn

    Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t * input_data

    Data to be encrypted. 32 bytes for Write command, 36 bytes for PrivWrite command.
- uint8_t * encrypted_data

    Encrypted version of input_data will be returned here. 32 bytes for Write command, 36 bytes for PrivWrite command.
- uint8_t * auth_mac

    Write MAC will be returned here. 32 bytes.
- struct atca_temp_key * temp_key

    Current state of TempKey.

9.48.1 Detailed Description

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

9.48.2 Field Documentation

9.48.2.1 auth_mac

```
uint8_t* auth_mac
```

Write MAC will be returned here. 32 bytes.

9.48.2.2 encrypted_data

```
uint8_t* encrypted_data
```

Encrypted version of input_data will be returned here. 32 bytes for Write command, 36 bytes for PrivWrite command.

9.48.2.3 input_data

```
const uint8_t* input_data
```

Data to be encrypted. 32 bytes for Write command, 36 bytes for PrivWrite command.

9.48.2.4 key_id

```
uint16_t key_id
```

KeyID/Param2 for the Write or PrivWrite command.

9.48.2.5 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

9.48.2.6 temp_key

```
struct atca_temp_key* temp_key
```

Current state of TempKey.

9.48.2.7 zone

```
uint8_t zone
```

Zone/Param1 for the Write or PrivWrite command.

9.49 atcacert_build_state_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- const atcacert_def_t * cert_def

  Certificate definition for the certificate being rebuilt.
- uint8_t * cert

  Buffer to contain the rebuilt certificate.
- size_t * cert_size

  Current size of the certificate in bytes.
- size_t max_cert_size

  Max size of the cert buffer in bytes.
- uint8_t is_device_sn

  Indicates the structure contains the device SN.
- uint8_t device_sn [9]

  Storage for the device SN, when it's found.

9.49.1 Detailed Description

Tracks the state of a certificate as it's being rebuilt from device information.

9.49.2 Field Documentation

9.49.2.1 cert

```
uint8_t* cert
```

Buffer to contain the rebuilt certificate.

9.49.2.2 cert_def

```
const atcacert_def_t* cert_def
```

Certificate definition for the certificate being rebuilt.

9.49.2.3 cert_size

```
size_t* cert_size
```

Current size of the certificate in bytes.

9.49.2.4 device_sn

```
uint8_t device_sn[9]
```

Storage for the device SN, when it's found.

9.49.2.5 is_device_sn

```
uint8_t is_device_sn
```

Indicates the structure contains the device SN.

9.49.2.6 max_cert_size

```
size_t max_cert_size
```

Max size of the cert buffer in bytes.

9.50 atcacert_cert_element_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- char id [25]

    ID identifying this element.
- atcacert_device_loc_t device_loc

    Location in the device for the element.
- atcacert_cert_loc_t cert_loc

    Location in the certificate template for the element.
- atcacert_transform_t transforms [2]

    List of transforms from device to cert for this element.

9.50.1 Detailed Description

Defines a generic dynamic element for a certificate including the device and template locations.

9.50.2 Field Documentation

9.50.2.1 cert_loc

```
atcacert_cert_loc_t cert_loc
```

Location in the certificate template for the element.

9.50.2.2 device_loc

```
atcacert_device_loc_t device_loc
```

Location in the device for the element.

9.50.2.3 id

```
char id[25]
```

ID identifying this element.

9.50.2.4 transforms

```
atcacert_transform_t transforms[2]
```

List of transforms from device to cert for this element.

9.51 atcacert_cert_loc_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- uint16_t offset

    Byte offset in the certificate template.
- uint16_t count

    Byte count. Set to 0 if it doesn't exist.

9.51.1 Detailed Description

Defines a chunk of data in a certificate template.

9.51.2 Field Documentation

9.51.2.1 count

```
uint16_t count
```

Byte count. Set to 0 if it doesn't exist.

9.51.2.2 offset

```
uint16_t offset
```

Byte offset in the certificate template.

9.52 atcacert_def_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- atcacert_cert_type_t type

  Certificate type.
- uint8_t template_id

  ID for the this certificate definition (4-bit value).
- uint8_t chain_id

  ID for the certificate chain this definition is a part of (4-bit value).
- uint8_t private_key_slot

  If this is a device certificate template, this is the device slot for the device private key.
- atcacert_cert_sn_src_t sn_source

  Where the certificate serial number comes from (4-bit value).
- atcacert_device_loc_t cert_sn_dev_loc

  Only applies when sn_source is SNSRC_STORED or SNSRC_STORED_DYNAMIC. Describes where to get the certificate serial number on the device.
- atcacert_date_format_t issue_date_format

  Format of the issue date in the certificate.
- atcacert_date_format_t expire_date_format

  format of the expire date in the certificate.
- atcacert_cert_loc_t tbs_cert_loc

  Location in the certificate for the TBS (to be signed) portion.
- uint8_t expire_years

  Number of years the certificate is valid for (5-bit value). 0 means no expiration.
- atcacert_device_loc_t public_key_dev_loc

  Where on the device the public key can be found.
- atcacert_device_loc_t comp_cert_dev_loc

  Where on the device the compressed cert can be found.
- atcacert_cert_loc_t std_cert_elements [STDCERT_NUM_ELEMENTS]

  Where in the certificate template the standard cert elements are inserted.
- const atcacert_cert_element_t * cert_elements

  Additional certificate elements outside of the standard certificate contents.
- uint8_t cert_elements_count

  Number of additional certificate elements in cert_elements.
- const uint8_t * cert_template

  Pointer to the actual certificate template data.
- uint16_t cert_template_size

  Size of the certificate template in cert_template in bytes.
- const struct atcacert_def_s * ca_cert_def

  Certificate definition of the CA certificate.

9.52.1 Detailed Description

Defines a certificate and all the pieces to work with it.

If any of the standard certificate elements (std_cert_elements) are not a part of the certificate definition, set their count to 0 to indicate their absence.

9.52.2 Field Documentation

9.52.2.1 ca_cert_def

```
const struct atcacert_def_s* ca_cert_def
```

Certificate definition of the CA certificate.

9.52.2.2 cert_elements

```
const atcacert_cert_element_t* cert_elements
```

Additional certificate elements outside of the standard certificate contents.

9.52.2.3 cert_elements_count

```
uint8_t cert_elements_count
```

Number of additional certificate elements in cert_elements.

9.52.2.4 cert_sn_dev_loc

```
atcacert_device_loc_t cert_sn_dev_loc
```

Only applies when sn_source is SNSRC_STORED or SNSRC_STORED_DYNAMIC. Describes where to get the certificate serial number on the device.

9.52.2.5 cert_template

```
const uint8_t* cert_template
```

Pointer to the actual certificate template data.

9.52.2.6 cert_template_size

```
uint16_t cert_template_size
```

Size of the certificate template in cert_template in bytes.

9.52.2.7 chain_id

```
uint8_t chain_id
```

ID for the certificate chain this definition is a part of (4-bit value).

9.52.2.8 comp_cert_dev_loc

```
atcacert_device_loc_t comp_cert_dev_loc
```

Where on the device the compressed cert can be found.

9.52.2.9 expire_date_format

```
atcacert_date_format_t expire_date_format
```

format of the expire date in the certificate.

9.52.2.10 expire_years

```
uint8_t expire_years
```

Number of years the certificate is valid for (5-bit value). 0 means no expiration.

9.52.2.11 issue_date_format

```
atcacert_date_format_t issue_date_format
```

Format of the issue date in the certificate.

9.52.2.12 private_key_slot

```
uint8_t private_key_slot
```

If this is a device certificate template, this is the device slot for the device private key.

9.52.2.13 public_key_dev_loc

```
atcacert_device_loc_t public_key_dev_loc
```

Where on the device the public key can be found.

9.52.2.14 sn_source

```
atcacert_cert_sn_src_t sn_source
```

Where the certificate serial number comes from (4-bit value).

9.52.2.15 std_cert_elements

```
atcacert_cert_loc_t std_cert_elements[STDCERT_NUM_ELEMENTS]
```

Where in the certificate template the standard cert elements are inserted.

9.52.2.16 tbs_cert_loc

```
atcacert_cert_loc_t tbs_cert_loc
```

Location in the certificate for the TBS (to be signed) portion.

9.52.2.17 template_id

```
uint8_t template_id
```

ID for the this certificate definition (4-bit value).

9.52.2.18 type

```
atcacert_cert_type_t type
```

Certificate type.

9.53 atcacert_device_loc_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- atcacert_device_zone_t zone

  Zone in the device.
- uint8_t slot

  Slot within the data zone. Only applies if zone is DEVZONE_DATA.
- uint8_t is_genkey

  If true, use GenKey command to get the contents instead of Read.
- uint16_t offset

  Byte offset in the zone.
- uint16_t count

  Byte count.

9.53.1 Detailed Description

Defines a chunk of data in an ATECC device.

9.53.2 Field Documentation

9.53.2.1 count

```
uint16_t count
```

Byte count.

9.53.2.2 is_genkey

```
uint8_t is_genkey
```

If true, use GenKey command to get the contents instead of Read.

9.53.2.3 offset

```
uint16_t offset
```

Byte offset in the zone.

9.53.2.4 slot

```
uint8_t slot
```

Slot within the data zone. Only applies if zone is DEVZONE_DATA.

9.53.2.5 zone

```
atcacert_device_zone_t zone
```

Zone in the device.

9.54 atcacert_tm_utc_s Struct Reference

```
#include <atcacert_date.h>
```

Data Fields

- int tm_sec
- int tm_min
- int tm_hour
- int tm_mday
- int tm_mon
- int tm_year

9.54.1 Detailed Description

Holds a broken-down date in UTC. Mimics atcacert_tm_utc_t from time.h.

9.54.2 Field Documentation

9.54.2.1 tm_hour

```
int tm_hour
```

9.54.2.2 tm_mday

```
int tm_mday
```

9.54.2.3 tm_min

```
int tm_min
```

9.54.2.4 tm_mon

```
int tm_mon
```

9.54.2.5 tm_sec

```
int tm_sec
```

9.54.2.6 tm_year

```
int tm_year
```

9.55 ATCAHAL_t Struct Reference

HAL Driver Structure.

```
#include <atca_iface.h>
```

Data Fields

- ATCA_STATUS(∗ halinit)(ATCAIface iface, ATCAIfaceCfg ∗cfg)
- ATCA_STATUS(∗ halpostinit)(ATCAIface iface)
- ATCA_STATUS(∗ halsend)(ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)
- ATCA_STATUS(∗ halreceive)(ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxlength)
- ATCA_STATUS(∗ halcontrol)(ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)
- ATCA_STATUS(∗ halrelease)(void ∗hal_data)

9.55.1 Detailed Description

HAL Driver Structure.

9.55.2 Field Documentation

9.55.2.1 halcontrol

```
ATCA_STATUS(* halcontrol) (ATCAIface iface, uint8_t option, void *param, size_t paramlen)
```

9.55.2.2 halinit

```
ATCA_STATUS(* halinit) (ATCAIface iface, ATCAIfaceCfg *cfg)
```

9.55.2.3 halpostinit

```
ATCA_STATUS(* halpostinit) (ATCAIface iface)
```

9.55.2.4 halreceive

```
ATCA_STATUS(* halreceive) (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_←
t *rxlength)
```

9.55.2.5 halrelease

```
ATCA_STATUS(* halrelease) (void *hal_data)
```

9.55.2.6 halsend

```
ATCA_STATUS(* halsend) (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)
```

9.56 atcaI2Cmaster Struct Reference

this is the hal_data for ATCA HAL for ASF SERCOM

```
#include <hal_uc3_i2c_asf.h>
```

Data Fields

- int id
- i2c_config_t conf
- int ref_ct
- uint8_t twi_id
- avr32_twi_t * twi_master_instance
- int bus_index

9.56.1 Detailed Description

this is the hal_data for ATCA HAL for ASF SERCOM

9.56.2 Field Documentation

9.56.2.1 bus_index

```
int bus_index
```

9.56.2.2 conf

```
i2c_config_t conf
```

9.56.2.3 id

```
int id
```

9.56.2.4 ref_ct

```
int ref_ct
```

9.56.2.5 twi_id

```
uint8_t twi_id
```

9.56.2.6 twi_master_instance

```
avr32_twi_t* twi_master_instance
```

9.57 ATCAIfaceCfg Struct Reference

```
#include <atca_iface.h>
```

Data Fields

- ATCAIfaceType iface_type
- ATCADeviceType devtype
- union {
  struct {
    uint8_t address
    uint8_t bus
    uint32_t baud
  } atcai2c
  struct {
    uint8_t address
    uint8_t bus
  } atcaswi
  struct {
    uint8_t bus
    uint8_t select_pin
    uint32_t baud
  } atcaspi
  struct {
    ATCAKitType dev_interface
    uint8_t dev_identity
    uint8_t port
    uint32_t baud
    uint8_t wordsize
    uint8_t parity
    uint8_t stopbits
  } atcauart
  struct {
    int idx
    ATCAKitType dev_interface
    uint8_t dev_identity
    uint32_t vid
    uint32_t pid
    uint32_t packetsize
  } atcahid
  struct {
    ATCAKitType dev_interface
    uint8_t dev_identity
    uint32_t flags
  } atcakit
  struct {
    ATCA_STATUS(∗ halinit)(void ∗hal, void ∗cfg)
    ATCA_STATUS(∗ halpostinit)(void ∗iface)
    ATCA_STATUS(∗ halsend)(void ∗iface, uint8_t
      word_address, uint8_t ∗txdata,

```
    int txlength)
  ATCA_STATUS(∗ halreceive )(void ∗iface, uint8_t
    word_address, uint8_t ∗rxdata,
    uint16_t ∗rxlength)
  ATCA_STATUS(∗ halwake )(void ∗iface)
          ATCA_STATUS(∗ halidle )(void ∗iface)
          ATCA_STATUS(∗ halsleep )(void ∗iface)
          ATCA_STATUS(∗ halrelease )(void ∗hal_data)
       } atcacustom
     };
```

- uint16_t wake_delay
- int rx_retries
- void ∗ cfg_data

9.57.1 Field Documentation

9.57.1.1 "@1

```
union { ... }
```

9.57.1.2 address

```
uint8_t address
```

Device address - the upper 7 bits are the I2c address bits

9.57.1.3 atcacustom

```
struct { ... } atcacustom
```

9.57.1.4 atcahid

```
struct { ... } atcahid
```

9.57.1.5 atcai2c

```
struct { ... } atcai2c
```

9.57.1.6 atcakit

```
struct { ... } atcakit
```

9.57.1.7 atcaspi

```
struct { ... } atcaspi
```

9.57.1.8 atcaswi

```
struct { ... } atcaswi
```

9.57.1.9 atcauart

```
struct { ... } atcauart
```

9.57.1.10 baud

```
uint32_t baud
```

9.57.1.11 bus

```
uint8_t bus
```

9.57.1.12 cfg_data

```
void* cfg_data
```

9.57.1.13 dev_identity

```
uint8_t dev_identity
```

9.57.1.14 dev_interface

```
ATCAKitType dev_interface
```

9.57.1.15 devtype

```
ATCADeviceType devtype
```

9.57.1.16 flags

```
uint32_t flags
```

9.57.1.17 halidle

```
ATCA_STATUS(* halidle) (void *iface)
```

9.57.1.18 halinit

```
ATCA_STATUS(* halinit) (void *hal, void *cfg)
```

9.57.1.19 halpostinit

```
ATCA_STATUS(* halpostinit) (void *iface)
```

9.57.1.20 halreceive

```
ATCA_STATUS(* halreceive) (void *iface, uint8_t word_address, uint8_t *rxdata, uint16_t *rxlength)
```

9.57.1.21 halrelease

```
ATCA_STATUS(* halrelease) (void *hal_data)
```

9.57.1.22 halsend

```
ATCA_STATUS(* halsend) (void *iface, uint8_t word_address, uint8_t *txdata, int txlength)
```

9.57.1.23 halsleep

```
ATCA_STATUS(* halsleep) (void *iface)
```

9.57.1.24 halwake

```
ATCA_STATUS(* halwake) (void *iface)
```

9.57.1.25 idx

```
int idx
```

9.57.1.26 iface_type

```
ATCAIfaceType iface_type
```

9.57.1.27 packetsize

```
uint32_t packetsize
```

9.57.1.28 parity

```
uint8_t parity
```

9.57.1.29 pid

```
uint32_t pid
```

9.57.1.30 port

```
uint8_t port
```

9.57.1.31 rx_retries

```
int rx_retries
```

9.57.1.32 select_pin

```
uint8_t select_pin
```

9.57.1.33 stopbits

```
uint8_t stopbits
```

9.57.1.34 vid

```
uint32_t vid
```

9.57.1.35 wake_delay

```
uint16_t wake_delay
```

9.57.1.36 wordsize

```
uint8_t wordsize
```

9.58 ATCAPacket Struct Reference

```
#include <calib_command.h>
```

Data Fields

- uint8_t _reserved
- uint8_t txsize
- uint8_t opcode
- uint8_t param1
- uint16_t param2
- uint8_t data [192]
- uint8_t execTime

9.58.1 Field Documentation

9.58.1.1 _reserved

```
uint8_t _reserved
```

9.58.1.2 data

```
uint8_t data[192]
```

9.58.1.3 execTime

```
uint8_t execTime
```

9.58.1.4 opcode

```
uint8_t opcode
```

9.58.1.5 param1

```
uint8_t param1
```

9.58.1.6 param2

```
uint16_t param2
```

9.58.1.7 txsize

```
uint8_t txsize
```

9.59 atcaSWImaster Struct Reference

this is the hal_data for ATCA HAL for ASF SERCOM

```
#include <swi_uart_samd21_asf.h>
```

Data Fields

- struct usart_module usart_instance
- int ref_ct
- int bus_index
- struct usart_sync_descriptor USART_SWI
- uint32_t sercom_core_freq

9.59.1 Detailed Description

this is the hal_data for ATCA HAL for ASF SERCOM

9.59.2 Field Documentation

9.59.2.1 bus_index

```
int bus_index
```

9.59.2.2 ref_ct

```
int ref_ct
```

9.59.2.3 sercom_core_freq

```
uint32_t sercom_core_freq
```

9.59.2.4 usart_instance

```
struct usart_module usart_instance
```

9.59.2.5 USART_SWI

```
struct usart_sync_descriptor USART_SWI
```

9.60 CK_AES_CBC_ENCRYPT_DATA_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE iv [16]
- CK_BYTE_PTR pData
- CK_ULONG length

9.60.1 Field Documentation

9.60.1.1 iv

```
CK_BYTE iv[16]
```

9.60.1.2 length

```
CK_ULONG length
```

9.60.1.3 pData

```
CK_BYTE_PTR pData
```

9.61 CK_AES_CCM_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulDataLen
- CK_BYTE_PTR pNonce
- CK_ULONG ulNonceLen
- CK_BYTE_PTR pAAD
- CK_ULONG ulAADLen
- CK_ULONG ulMACLen

9.61.1 Field Documentation

9.61.1.1 pAAD

```
CK_BYTE_PTR pAAD
```

9.61.1.2 pNonce

```
CK_BYTE_PTR pNonce
```

9.61.1.3 ulAADLen

```
CK_ULONG ulAADLen
```

9.61.1.4 ulDataLen

```
CK_ULONG ulDataLen
```

9.61.1.5 ulMACLen

```
CK_ULONG ulMACLen
```

9.61.1.6 ulNonceLen

```
CK_ULONG ulNonceLen
```

9.62 CK_AES_CTR_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulCounterBits
- CK_BYTE cb [16]

9.62.1 Field Documentation

9.62.1.1 cb

```
CK_BYTE cb[16]
```

9.62.1.2 ulCounterBits

```
CK_ULONG ulCounterBits
```

9.63 CK_AES_GCM_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pIv
- CK_ULONG ulIvLen
- CK_ULONG ulIvBits
- CK_BYTE_PTR pAAD
- CK_ULONG ulAADLen
- CK_ULONG ulTagBits

9.63.1 Field Documentation

9.63.1.1 pAAD

```
CK_BYTE_PTR pAAD
```

9.63.1.2 pIv

```
CK_BYTE_PTR pIv
```

9.63.1.3 ulAADLen

```
CK_ULONG ulAADLen
```

9.63.1.4 ulIvBits

```
CK_ULONG ulIvBits
```

9.63.1.5 ulIvLen

```
CK_ULONG ulIvLen
```

9.63.1.6 ulTagBits

```
CK_ULONG ulTagBits
```

9.64 CK_ARIA_CBC_ENCRYPT_DATA_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE iv [16]
- CK_BYTE_PTR pData
- CK_ULONG length

9.64.1 Field Documentation

9.64.1.1 iv

```
CK_BYTE iv[16]
```

9.64.1.2 length

```
CK_ULONG length
```

9.64.1.3 pData

```
CK_BYTE_PTR pData
```

9.65 CK_ATTRIBUTE Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ATTRIBUTE_TYPE type
- CK_VOID_PTR pValue
- CK_ULONG ulValueLen

9.65.1 Field Documentation

9.65.1.1 pValue

```
CK_VOID_PTR pValue
```

9.65.1.2 type

```
CK_ATTRIBUTE_TYPE type
```

9.65.1.3 ulValueLen

```
CK_ULONG ulValueLen
```

9.66 CK_C_INITIALIZE_ARGS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_CREATEMUTEX CreateMutex
- CK_DESTROYMUTEX DestroyMutex
- CK_LOCKMUTEX LockMutex
- CK_UNLOCKMUTEX UnlockMutex
- CK_FLAGS flags
- CK_VOID_PTR pReserved

9.66.1 Field Documentation

9.66.1.1 CreateMutex

```
CK_CREATEMUTEX CreateMutex
```

9.66.1.2 DestroyMutex

```
CK_DESTROYMUTEX DestroyMutex
```

9.66.1.3 flags

```
CK_FLAGS flags
```

9.66.1.4 LockMutex

```
CK_LOCKMUTEX LockMutex
```

9.66.1.5 pReserved

```
CK_VOID_PTR pReserved
```

9.66.1.6 UnlockMutex

```
CK_UNLOCKMUTEX UnlockMutex
```

9.67 CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE iv [16]
- CK_BYTE_PTR pData
- CK_ULONG length

9.67.1 Field Documentation

9.67.1.1 iv

```
CK_BYTE iv[16]
```

9.67.1.2 length

```
CK_ULONG length
```

9.67.1.3 pData

```
CK_BYTE_PTR pData
```

9.68 CK_CAMELLIA_CTR_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulCounterBits
- CK_BYTE cb [16]

9.68.1 Field Documentation

9.68.1.1 cb

```
CK_BYTE cb[16]
```

9.68.1.2 ulCounterBits

```
CK_ULONG ulCounterBits
```

9.69 CK_CCM_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulDataLen
- CK_BYTE_PTR pNonce
- CK_ULONG ulNonceLen
- CK_BYTE_PTR pAAD
- CK_ULONG ulAADLen
- CK_ULONG ulMACLen

9.69.1 Field Documentation

9.69.1.1 pAAD

```
CK_BYTE_PTR pAAD
```

9.69.1.2 pNonce

```
CK_BYTE_PTR pNonce
```

9.69.1.3 ulAADLen

```
CK_ULONG ulAADLen
```

9.69.1.4 ulDataLen

```
CK_ULONG ulDataLen
```

9.69.1.5 ulMACLen

```
CK_ULONG ulMACLen
```

9.69.1.6 ulNonceLen

```
CK_ULONG ulNonceLen
```

9.70 CK_CMS_SIG_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OBJECT_HANDLE certificateHandle
- CK_MECHANISM_PTR pSigningMechanism
- CK_MECHANISM_PTR pDigestMechanism
- CK_UTF8CHAR_PTR pContentType
- CK_BYTE_PTR pRequestedAttributes
- CK_ULONG ulRequestedAttributesLen
- CK_BYTE_PTR pRequiredAttributes
- CK_ULONG ulRequiredAttributesLen

9.70.1 Field Documentation

9.70.1.1 certificateHandle

```
CK_OBJECT_HANDLE certificateHandle
```

9.70.1.2 pContentType

```
CK_UTF8CHAR_PTR pContentType
```

9.70.1.3 pDigestMechanism

```
CK_MECHANISM_PTR pDigestMechanism
```

9.70.1.4 pRequestedAttributes

```
CK_BYTE_PTR pRequestedAttributes
```

9.70.1.5 pRequiredAttributes

```
CK_BYTE_PTR pRequiredAttributes
```

9.70.1.6 pSigningMechanism

```
CK_MECHANISM_PTR pSigningMechanism
```

9.70.1.7 ulRequestedAttributesLen

```
CK_ULONG ulRequestedAttributesLen
```

9.70.1.8 ulRequiredAttributesLen

```
CK_ULONG ulRequiredAttributesLen
```

9.71 CK_DATE Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_CHAR year [4]
- CK_CHAR month [2]
- CK_CHAR day [2]

9.71.1 Field Documentation

9.71.1.1 day

```
CK_CHAR day[2]
```

9.71.1.2 month

```
CK_CHAR month[2]
```

9.71.1.3 year

```
CK_CHAR year[4]
```

9.72 CK_DES_CBC_ENCRYPT_DATA_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE iv [8]
- CK_BYTE_PTR pData
- CK_ULONG length

9.72.1 Field Documentation

9.72.1.1 iv

```
CK_BYTE iv[8]
```

9.72.1.2 length

```
CK_ULONG length
```

9.72.1.3 pData

```
CK_BYTE_PTR pData
```

9.73 CK_DSA_PARAMETER_GEN_PARAM Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE hash
- CK_BYTE_PTR pSeed
- CK_ULONG ulSeedLen
- CK_ULONG ulIndex

9.73.1 Field Documentation

9.73.1.1 hash

```
CK_MECHANISM_TYPE hash
```

9.73.1.2 pSeed

```
CK_BYTE_PTR pSeed
```

9.73.1.3 ulIndex

```
CK_ULONG ulIndex
```

9.73.1.4 ulSeedLen

```
CK_ULONG ulSeedLen
```

9.74 CK_ECDH1_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_EC_KDF_TYPE kdf
- CK_ULONG ulSharedDataLen
- CK_BYTE_PTR pSharedData
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData

9.74.1 Field Documentation

9.74.1.1 kdf

```
CK_EC_KDF_TYPE kdf
```

9.74.1.2 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.74.1.3 pSharedData

```
CK_BYTE_PTR pSharedData
```

9.74.1.4 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.74.1.5 ulSharedDataLen

```
CK_ULONG ulSharedDataLen
```

9.75 CK_ECDH2_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_EC_KDF_TYPE kdf
- CK_ULONG ulSharedDataLen
- CK_BYTE_PTR pSharedData
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPrivateDataLen
- CK_OBJECT_HANDLE hPrivateData
- CK_ULONG ulPublicDataLen2
- CK_BYTE_PTR pPublicData2

9.75.1 Field Documentation

9.75.1.1 hPrivateData

```
CK_OBJECT_HANDLE hPrivateData
```

9.75.1.2 kdf

```
CK_EC_KDF_TYPE kdf
```

9.75.1.3 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.75.1.4 pPublicData2

```
CK_BYTE_PTR pPublicData2
```

9.75.1.5 pSharedData

```
CK_BYTE_PTR pSharedData
```

9.75.1.6 ulPrivateDataLen

```
CK_ULONG ulPrivateDataLen
```

9.75.1.7 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.75.1.8 ulPublicDataLen2

```
CK_ULONG ulPublicDataLen2
```

9.75.1.9 ulSharedDataLen

```
CK_ULONG ulSharedDataLen
```

9.76 CK_ECDH_AES_KEY_WRAP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulAESKeyBits
- CK_EC_KDF_TYPE kdf
- CK_ULONG ulSharedDataLen
- CK_BYTE_PTR pSharedData

9.76.1 Field Documentation

9.76.1.1 kdf

```
CK_EC_KDF_TYPE kdf
```

9.76.1.2 pSharedData

```
CK_BYTE_PTR pSharedData
```

9.76.1.3 ulAESKeyBits

```
CK_ULONG ulAESKeyBits
```

9.76.1.4 ulSharedDataLen

```
CK_ULONG ulSharedDataLen
```

9.77 CK_ECMQV_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_EC_KDF_TYPE kdf
- CK_ULONG ulSharedDataLen
- CK_BYTE_PTR pSharedData
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPrivateDataLen
- CK_OBJECT_HANDLE hPrivateData
- CK_ULONG ulPublicDataLen2
- CK_BYTE_PTR pPublicData2
- CK_OBJECT_HANDLE publicKey

9.77.1 Field Documentation

9.77.1.1 hPrivateData

```
CK_OBJECT_HANDLE hPrivateData
```

9.77.1.2 kdf

```
CK_EC_KDF_TYPE kdf
```

9.77.1.3 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.77.1.4 pPublicData2

```
CK_BYTE_PTR pPublicData2
```

9.77.1.5 pSharedData

```
CK_BYTE_PTR pSharedData
```

9.77.1.6 publicKey

```
CK_OBJECT_HANDLE publicKey
```

9.77.1.7 ulPrivateDataLen

```
CK_ULONG ulPrivateDataLen
```

9.77.1.8 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.77.1.9 ulPublicDataLen2

```
CK_ULONG ulPublicDataLen2
```

9.77.1.10 ulSharedDataLen

```
CK_ULONG ulSharedDataLen
```

9.78 CK_FUNCTION_LIST Struct Reference

```
#include <pkcs11.h>
```

Data Fields

- CK_VERSION version

9.78.1 Field Documentation

9.78.1.1 version

```
CK_VERSION version
```

9.79 CK_GCM_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pIv
- CK_ULONG ulIvLen
- CK_ULONG ulIvBits
- CK_BYTE_PTR pAAD
- CK_ULONG ulAADLen
- CK_ULONG ulTagBits

9.79.1 Field Documentation

9.79.1.1 pAAD

```
CK_BYTE_PTR pAAD
```

9.79.1.2 pIv

```
CK_BYTE_PTR pIv
```

9.79.1.3 ulAADLen

```
CK_ULONG ulAADLen
```

9.79.1.4 ulIvBits

```
CK_ULONG ulIvBits
```

9.79.1.5 ulIvLen

```
CK_ULONG ulIvLen
```

9.79.1.6 ulTagBits

```
CK_ULONG ulTagBits
```

9.80 CK_GOSTR3410_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_EC_KDF_TYPE kdf
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pUKM
- CK_ULONG ulUKMLen

9.80.1 Field Documentation

9.80.1.1 kdf

```
CK_EC_KDF_TYPE kdf
```

9.80.1.2 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.80.1.3 pUKM

```
CK_BYTE_PTR pUKM
```

9.80.1.4 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.80.1.5 ulUKMLen

```
CK_ULONG ulUKMLen
```

9.81 CK_GOSTR3410_KEY_WRAP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pWrapOID
- CK_ULONG ulWrapOIDLen
- CK_BYTE_PTR pUKM
- CK_ULONG ulUKMLen
- CK_OBJECT_HANDLE hKey

9.81.1 Field Documentation

9.81.1.1 hKey

```
CK_OBJECT_HANDLE hKey
```

9.81.1.2 pUKM

```
CK_BYTE_PTR pUKM
```

9.81.1.3 pWrapOID

```
CK_BYTE_PTR pWrapOID
```

9.81.1.4 ulUKMLen

```
CK_ULONG ulUKMLen
```

9.81.1.5 ulWrapOIDLen

```
CK_ULONG ulWrapOIDLen
```

9.82 CK_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_VERSION cryptokiVersion
- CK_UTF8CHAR manufacturerID [32]
- CK_FLAGS flags
- CK_UTF8CHAR libraryDescription [32]
- CK_VERSION libraryVersion

9.82.1 Field Documentation

9.82.1.1 cryptokiVersion

```
CK_VERSION cryptokiVersion
```

9.82.1.2 flags

```
CK_FLAGS flags
```

9.82.1.3 libraryDescription

```
CK_UTF8CHAR libraryDescription[32]
```

9.82.1.4 libraryVersion

```
CK_VERSION libraryVersion
```

9.82.1.5 manufacturerID

```
CK_UTF8CHAR manufacturerID[32]
```

9.83 CK_KEA_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BBOOL isSender
- CK_ULONG ulRandomLen
- CK_BYTE_PTR pRandomA
- CK_BYTE_PTR pRandomB
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData

9.83.1 Field Documentation

9.83.1.1 isSender

```
CK_BBOOL isSender
```

9.83.1.2 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.83.1.3 pRandomA

```
CK_BYTE_PTR pRandomA
```

9.83.1.4 pRandomB

```
CK_BYTE_PTR pRandomB
```

9.83.1.5 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.83.1.6 ulRandomLen

```
CK_ULONG ulRandomLen
```

9.84 CK_KEY_DERIVATION_STRING_DATA Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pData
- CK_ULONG ulLen

9.84.1 Field Documentation

9.84.1.1 pData

```
CK_BYTE_PTR pData
```

9.84.1.2 ulLen

```
CK_ULONG ulLen
```

9.85 CK_KEY_WRAP_SET_OAEP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE bBC
- CK_BYTE_PTR pX
- CK_ULONG ulXLen

9.85.1 Field Documentation

9.85.1.1 bBC

```
CK_BYTE bBC
```

9.85.1.2 pX

```
CK_BYTE_PTR pX
```

9.85.1.3 ulXLen

```
CK_ULONG ulXLen
```

9.86 CK_KIP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_PTR pMechanism
- CK_OBJECT_HANDLE hKey
- CK_BYTE_PTR pSeed
- CK_ULONG ulSeedLen

9.86.1 Field Documentation

9.86.1.1 hKey

```
CK_OBJECT_HANDLE hKey
```

9.86.1.2 pMechanism

```
CK_MECHANISM_PTR pMechanism
```

9.86.1.3 pSeed

```
CK_BYTE_PTR pSeed
```

9.86.1.4 ulSeedLen

```
CK_ULONG ulSeedLen
```

9.87 CK_MECHANISM Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE mechanism
- CK_VOID_PTR pParameter
- CK_ULONG ulParameterLen

9.87.1 Field Documentation

9.87.1.1 mechanism

```
CK_MECHANISM_TYPE mechanism
```

9.87.1.2 pParameter

```
CK_VOID_PTR pParameter
```

9.87.1.3 ulParameterLen

```
CK_ULONG ulParameterLen
```

9.88 CK_MECHANISM_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulMinKeySize
- CK_ULONG ulMaxKeySize
- CK_FLAGS flags

9.88.1 Field Documentation

9.88.1.1 flags

```
CK_FLAGS flags
```

9.88.1.2 ulMaxKeySize

```
CK_ULONG ulMaxKeySize
```

9.88.1.3 ulMinKeySize

```
CK_ULONG ulMinKeySize
```

9.89 CK_OTP_PARAM Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OTP_PARAM_TYPE type
- CK_VOID_PTR pValue
- CK_ULONG ulValueLen

9.89.1 Field Documentation

9.89.1.1 pValue

```
CK_VOID_PTR pValue
```

9.89.1.2 type

```
CK_OTP_PARAM_TYPE type
```

9.89.1.3 ulValueLen

```
CK_ULONG ulValueLen
```

9.90 CK_OTP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OTP_PARAM_PTR pParams
- CK_ULONG ulCount

9.90.1 Field Documentation

9.90.1.1 pParams

```
CK_OTP_PARAM_PTR pParams
```

9.90.1.2 ulCount

```
CK_ULONG ulCount
```

9.91 CK_OTP_SIGNATURE_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OTP_PARAM_PTR pParams
- CK_ULONG ulCount

9.91.1 Field Documentation

9.91.1.1 pParams

```
CK_OTP_PARAM_PTR pParams
```

9.91.1.2 ulCount

```
CK_ULONG ulCount
```

9.92 CK_PBE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pInitVector
- CK_UTF8CHAR_PTR pPassword
- CK_ULONG ulPasswordLen
- CK_BYTE_PTR pSalt
- CK_ULONG ulSaltLen
- CK_ULONG ulIteration

9.92.1 Field Documentation

9.92.1.1 pInitVector

```
CK_BYTE_PTR pInitVector
```

9.92.1.2 pPassword

```
CK_UTF8CHAR_PTR pPassword
```

9.92.1.3 pSalt

```
CK_BYTE_PTR pSalt
```

9.92.1.4 ulIteration

```
CK_ULONG ulIteration
```

9.92.1.5 ulPasswordLen

```
CK_ULONG ulPasswordLen
```

9.92.1.6 ulSaltLen

```
CK_ULONG ulSaltLen
```

9.93 CK_PKCS5_PBKD2_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource
- CK_VOID_PTR pSaltSourceData
- CK_ULONG ulSaltSourceDataLen
- CK_ULONG iterations
- CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf
- CK_VOID_PTR pPrfData
- CK_ULONG ulPrfDataLen
- CK_UTF8CHAR_PTR pPassword
- CK_ULONG_PTR ulPasswordLen

9.93.1 Field Documentation

9.93.1.1 iterations

```
CK_ULONG iterations
```

9.93.1.2 pPassword

```
CK_UTF8CHAR_PTR pPassword
```

9.93.1.3 pPrfData

```
CK_VOID_PTR pPrfData
```

9.93.1.4 prf

```
CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf
```

9.93.1.5 pSaltSourceData

```
CK_VOID_PTR pSaltSourceData
```

9.93.1.6 saltSource

```
CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource
```

9.93.1.7 ulPasswordLen

```
CK_ULONG_PTR ulPasswordLen
```

9.93.1.8 ulPrfDataLen

```
CK_ULONG ulPrfDataLen
```

9.93.1.9 ulSaltSourceDataLen

```
CK_ULONG ulSaltSourceDataLen
```

9.94 CK_PKCS5_PBKD2_PARAMS2 Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource
- CK_VOID_PTR pSaltSourceData
- CK_ULONG ulSaltSourceDataLen
- CK_ULONG iterations
- CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf
- CK_VOID_PTR pPrfData
- CK_ULONG ulPrfDataLen
- CK_UTF8CHAR_PTR pPassword
- CK_ULONG ulPasswordLen

9.94.1 Field Documentation

9.94.1.1 iterations

```
CK_ULONG iterations
```

9.94.1.2 pPassword

```
CK_UTF8CHAR_PTR pPassword
```

9.94.1.3 pPrfData

```
CK_VOID_PTR pPrfData
```

9.94.1.4 prf

```
CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf
```

9.94.1.5 pSaltSourceData

```
CK_VOID_PTR pSaltSourceData
```

9.94.1.6 saltSource

```
CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource
```

9.94.1.7 ulPasswordLen

```
CK_ULONG ulPasswordLen
```

9.94.1.8 ulPrfDataLen

```
CK_ULONG ulPrfDataLen
```

9.94.1.9 ulSaltSourceDataLen

```
CK_ULONG ulSaltSourceDataLen
```

9.95 CK_RC2_CBC_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulEffectiveBits
- CK_BYTE iv [8]

9.95.1 Field Documentation

9.95.1.1 iv

```
CK_BYTE iv[8]
```

9.95.1.2 ulEffectiveBits

```
CK_ULONG ulEffectiveBits
```

9.96 CK_RC2_MAC_GENERAL_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulEffectiveBits
- CK_ULONG ulMacLength

9.96.1 Field Documentation

9.96.1.1 ulEffectiveBits

```
CK_ULONG ulEffectiveBits
```

9.96.1.2 ulMacLength

```
CK_ULONG ulMacLength
```

9.97 CK_RC5_CBC_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulWordsize
- CK_ULONG ulRounds
- CK_BYTE_PTR pIv
- CK_ULONG ulIvLen

9.97.1 Field Documentation

9.97.1.1 pIv

```
CK_BYTE_PTR pIv
```

9.97.1.2 ulIvLen

```
CK_ULONG ulIvLen
```

9.97.1.3 ulRounds

```
CK_ULONG ulRounds
```

9.97.1.4 ulWordsize

```
CK_ULONG ulWordsize
```

9.98 CK_RC5_MAC_GENERAL_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulWordsize
- CK_ULONG ulRounds
- CK_ULONG ulMacLength

9.98.1 Field Documentation

9.98.1.1 ulMacLength

```
CK_ULONG ulMacLength
```

9.98.1.2 ulRounds

```
CK_ULONG ulRounds
```

9.98.1.3 ulWordsize

```
CK_ULONG ulWordsize
```

9.99 CK_RC5_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulWordsize
- CK_ULONG ulRounds

9.99.1 Field Documentation

9.99.1.1 ulRounds

```
CK_ULONG ulRounds
```

9.99.1.2 ulWordsize

```
CK_ULONG ulWordsize
```

9.100 CK_RSA_AES_KEY_WRAP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulAESKeyBits
- CK_RSA_PKCS_OAEP_PARAMS_PTR pOAEPParams

9.100.1 Field Documentation

9.100.1.1 pOAEPParams

```
CK_RSA_PKCS_OAEP_PARAMS_PTR pOAEPParams
```

9.100.1.2 ulAESKeyBits

```
CK_ULONG ulAESKeyBits
```

9.101 CK_RSA_PKCS_OAEP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE hashAlg
- CK_RSA_PKCS_MGF_TYPE mgf
- CK_RSA_PKCS_OAEP_SOURCE_TYPE source
- CK_VOID_PTR pSourceData
- CK_ULONG ulSourceDataLen

9.101.1 Field Documentation

9.101.1.1 hashAlg

```
CK_MECHANISM_TYPE hashAlg
```

9.101.1.2 mgf

```
CK_RSA_PKCS_MGF_TYPE mgf
```

9.101.1.3 pSourceData

```
CK_VOID_PTR pSourceData
```

9.101.1.4 source

```
CK_RSA_PKCS_OAEP_SOURCE_TYPE source
```

9.101.1.5 ulSourceDataLen

```
CK_ULONG ulSourceDataLen
```

9.102 CK_RSA_PKCS_PSS_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE hashAlg
- CK_RSA_PKCS_MGF_TYPE mgf
- CK_ULONG sLen

9.102.1 Field Documentation

9.102.1.1 hashAlg

```
CK_MECHANISM_TYPE hashAlg
```

9.102.1.2 mgf

```
CK_RSA_PKCS_MGF_TYPE mgf
```

9.102.1.3 sLen

```
CK_ULONG sLen
```

9.103 CK_SEED_CBC_ENCRYPT_DATA_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE iv [16]
- CK_BYTE_PTR pData
- CK_ULONG length

9.103.1 Field Documentation

9.103.1.1 iv

```
CK_BYTE iv[16]
```

9.103.1.2 length

```
CK_ULONG length
```

9.103.1.3 pData

```
CK_BYTE_PTR pData
```

9.104 CK_SESSION_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_SLOT_ID slotID
- CK_STATE state
- CK_FLAGS flags
- CK_ULONG ulDeviceError

9.104.1 Field Documentation

9.104.1.1 flags

```
CK_FLAGS flags
```

9.104.1.2 slotID

```
CK_SLOT_ID slotID
```

9.104.1.3 state

```
CK_STATE state
```

9.104.1.4 ulDeviceError

```
CK_ULONG ulDeviceError
```

9.105 CK_SKIPJACK_PRIVATE_WRAP_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulPasswordLen
- CK_BYTE_PTR pPassword
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPAndGLen
- CK_ULONG ulQLen
- CK_ULONG ulRandomLen
- CK_BYTE_PTR pRandomA
- CK_BYTE_PTR pPrimeP
- CK_BYTE_PTR pBaseG
- CK_BYTE_PTR pSubprimeQ

9.105.1 Field Documentation

9.105.1.1 pBaseG

```
CK_BYTE_PTR pBaseG
```

9.105.1.2 pPassword

```
CK_BYTE_PTR pPassword
```

9.105.1.3 pPrimeP

```
CK_BYTE_PTR pPrimeP
```

9.105.1.4 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.105.1.5 pRandomA

```
CK_BYTE_PTR pRandomA
```

9.105.1.6 pSubprimeQ

```
CK_BYTE_PTR pSubprimeQ
```

9.105.1.7 ulPAndGLen

```
CK_ULONG ulPAndGLen
```

9.105.1.8 ulPasswordLen

```
CK_ULONG ulPasswordLen
```

9.105.1.9 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.105.1.10 ulQLen

```
CK_ULONG ulQLen
```

9.105.1.11 ulRandomLen

```
CK_ULONG ulRandomLen
```

9.106 CK_SKIPJACK_RELAYX_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulOldWrappedXLen
- CK_BYTE_PTR pOldWrappedX
- CK_ULONG ulOldPasswordLen
- CK_BYTE_PTR pOldPassword
- CK_ULONG ulOldPublicDataLen
- CK_BYTE_PTR pOldPublicData
- CK_ULONG ulOldRandomLen
- CK_BYTE_PTR pOldRandomA
- CK_ULONG ulNewPasswordLen
- CK_BYTE_PTR pNewPassword
- CK_ULONG ulNewPublicDataLen
- CK_BYTE_PTR pNewPublicData
- CK_ULONG ulNewRandomLen
- CK_BYTE_PTR pNewRandomA

9.106.1 Field Documentation

9.106.1.1 pNewPassword

```
CK_BYTE_PTR pNewPassword
```

9.106.1.2 pNewPublicData

```
CK_BYTE_PTR pNewPublicData
```

9.106.1.3 pNewRandomA

```
CK_BYTE_PTR pNewRandomA
```

9.106.1.4 pOldPassword

```
CK_BYTE_PTR pOldPassword
```

9.106.1.5 pOldPublicData

```
CK_BYTE_PTR pOldPublicData
```

9.106.1.6 pOldRandomA

```
CK_BYTE_PTR pOldRandomA
```

9.106.1.7 pOldWrappedX

```
CK_BYTE_PTR pOldWrappedX
```

9.106.1.8 ulNewPasswordLen

```
CK_ULONG ulNewPasswordLen
```

9.106.1.9 ulNewPublicDataLen

```
CK_ULONG ulNewPublicDataLen
```

9.106.1.10 ulNewRandomLen

```
CK_ULONG ulNewRandomLen
```

9.106.1.11 ulOldPasswordLen

```
CK_ULONG ulOldPasswordLen
```

9.106.1.12 ulOldPublicDataLen

```
CK_ULONG ulOldPublicDataLen
```

9.106.1.13 ulOldRandomLen

```
CK_ULONG ulOldRandomLen
```

9.106.1.14 ulOldWrappedXLen

```
CK_ULONG ulOldWrappedXLen
```

9.107 CK_SLOT_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_UTF8CHAR slotDescription [64]
- CK_UTF8CHAR manufacturerID [32]
- CK_FLAGS flags
- CK_VERSION hardwareVersion
- CK_VERSION firmwareVersion

9.107.1 Field Documentation

9.107.1.1 firmwareVersion

```
CK_VERSION firmwareVersion
```

9.107.1.2 flags

```
CK_FLAGS flags
```

9.107.1.3 hardwareVersion

```
CK_VERSION hardwareVersion
```

9.107.1.4 manufacturerID

```
CK_UTF8CHAR manufacturerID[32]
```

9.107.1.5 slotDescription

```
CK_UTF8CHAR slotDescription[64]
```

9.108 CK_SSL3_KEY_MAT_OUT Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OBJECT_HANDLE hClientMacSecret
- CK_OBJECT_HANDLE hServerMacSecret
- CK_OBJECT_HANDLE hClientKey
- CK_OBJECT_HANDLE hServerKey
- CK_BYTE_PTR pIVClient
- CK_BYTE_PTR pIVServer

9.108.1 Field Documentation

9.108.1.1 hClientKey

```
CK_OBJECT_HANDLE hClientKey
```

9.108.1.2 hClientMacSecret

```
CK_OBJECT_HANDLE hClientMacSecret
```

9.108.1.3 hServerKey

```
CK_OBJECT_HANDLE hServerKey
```

9.108.1.4 hServerMacSecret

```
CK_OBJECT_HANDLE hServerMacSecret
```

9.108.1.5 pIVClient

```
CK_BYTE_PTR pIVClient
```

9.108.1.6 pIVServer

```
CK_BYTE_PTR pIVServer
```

9.109 CK_SSL3_KEY_MAT_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulMacSizeInBits
- CK_ULONG ulKeySizeInBits
- CK_ULONG ulIVSizeInBits
- CK_BBOOL bIsExport
- CK_SSL3_RANDOM_DATA RandomInfo
- CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial

9.109.1 Field Documentation

9.109.1.1 bIsExport

```
CK_BBOOL bIsExport
```

9.109.1.2 pReturnedKeyMaterial

```
CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial
```

9.109.1.3 RandomInfo

```
CK_SSL3_RANDOM_DATA RandomInfo
```

9.109.1.4 ulIVSizeInBits

```
CK_ULONG ulIVSizeInBits
```

9.109.1.5 ulKeySizeInBits

```
CK_ULONG ulKeySizeInBits
```

9.109.1.6 ulMacSizeInBits

```
CK_ULONG ulMacSizeInBits
```

9.110 CK_SSL3_MASTER_KEY_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_SSL3_RANDOM_DATA RandomInfo
- CK_VERSION_PTR pVersion

9.110.1 Field Documentation

9.110.1.1 pVersion

```
CK_VERSION_PTR pVersion
```

9.110.1.2 RandomInfo

```
CK_SSL3_RANDOM_DATA RandomInfo
```

9.111 CK_SSL3_RANDOM_DATA Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pClientRandom
- CK_ULONG ulClientRandomLen
- CK_BYTE_PTR pServerRandom
- CK_ULONG ulServerRandomLen

9.111.1 Field Documentation

9.111.1.1 pClientRandom

```
CK_BYTE_PTR pClientRandom
```

9.111.1.2 pServerRandom

```
CK_BYTE_PTR pServerRandom
```

9.111.1.3 ulClientRandomLen

```
CK_ULONG ulClientRandomLen
```

9.111.1.4 ulServerRandomLen

```
CK_ULONG ulServerRandomLen
```

9.112 CK_TLS12_KEY_MAT_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_ULONG ulMacSizeInBits
- CK_ULONG ulKeySizeInBits
- CK_ULONG ulIVSizeInBits
- CK_BBOOL bIsExport
- CK_SSL3_RANDOM_DATA RandomInfo
- CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial
- CK_MECHANISM_TYPE prfHashMechanism

9.112.1 Field Documentation

9.112.1.1 bIsExport

```
CK_BBOOL bIsExport
```

9.112.1.2 pReturnedKeyMaterial

```
CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial
```

9.112.1.3 prfHashMechanism

```
CK_MECHANISM_TYPE prfHashMechanism
```

9.112.1.4 RandomInfo

```
CK_SSL3_RANDOM_DATA RandomInfo
```

9.112.1.5 ulIVSizeInBits

```
CK_ULONG ulIVSizeInBits
```

9.112.1.6 ulKeySizeInBits

```
CK_ULONG ulKeySizeInBits
```

9.112.1.7 ulMacSizeInBits

```
CK_ULONG ulMacSizeInBits
```

9.113 CK_TLS12_MASTER_KEY_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_SSL3_RANDOM_DATA RandomInfo
- CK_VERSION_PTR pVersion
- CK_MECHANISM_TYPE prfHashMechanism

9.113.1 Field Documentation

9.113.1.1 prfHashMechanism

```
CK_MECHANISM_TYPE prfHashMechanism
```

9.113.1.2 pVersion

```
CK_VERSION_PTR pVersion
```

9.113.1.3 RandomInfo

```
CK_SSL3_RANDOM_DATA RandomInfo
```

9.114 CK_TLS_KDF_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE prfMechanism
- CK_BYTE_PTR pLabel
- CK_ULONG ulLabelLength
- CK_SSL3_RANDOM_DATA RandomInfo
- CK_BYTE_PTR pContextData
- CK_ULONG ulContextDataLength

9.114.1 Field Documentation

9.114.1.1 pContextData

```
CK_BYTE_PTR pContextData
```

9.114.1.2 pLabel

```
CK_BYTE_PTR pLabel
```

9.114.1.3 prfMechanism

```
CK_MECHANISM_TYPE prfMechanism
```

9.114.1.4 RandomInfo

```
CK_SSL3_RANDOM_DATA RandomInfo
```

9.114.1.5 ulContextDataLength

```
CK_ULONG ulContextDataLength
```

9.114.1.6 ulLabelLength

```
CK_ULONG ulLabelLength
```

9.115 CK_TLS_MAC_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE prfHashMechanism
- CK_ULONG ulMacLength
- CK_ULONG ulServerOrClient

9.115.1 Field Documentation

9.115.1.1 prfHashMechanism

```
CK_MECHANISM_TYPE prfHashMechanism
```

9.115.1.2 ulMacLength

```
CK_ULONG ulMacLength
```

9.115.1.3 ulServerOrClient

```
CK_ULONG ulServerOrClient
```

9.116 CK_TLS_PRF_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pSeed
- CK_ULONG ulSeedLen
- CK_BYTE_PTR pLabel
- CK_ULONG ulLabelLen
- CK_BYTE_PTR pOutput
- CK_ULONG_PTR pulOutputLen

9.116.1 Field Documentation

9.116.1.1 pLabel

```
CK_BYTE_PTR pLabel
```

9.116.1.2 pOutput

```
CK_BYTE_PTR pOutput
```

9.116.1.3 pSeed

```
CK_BYTE_PTR pSeed
```

9.116.1.4 pulOutputLen

```
CK_ULONG_PTR pulOutputLen
```

9.116.1.5 ulLabelLen

```
CK_ULONG ulLabelLen
```

9.116.1.6 ulSeedLen

```
CK_ULONG ulSeedLen
```

9.117 CK_TOKEN_INFO Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_UTF8CHAR label [32]
- CK_UTF8CHAR manufacturerID [32]
- CK_UTF8CHAR model [16]
- CK_CHAR serialNumber [16]
- CK_FLAGS flags
- CK_ULONG ulMaxSessionCount
- CK_ULONG ulSessionCount
- CK_ULONG ulMaxRwSessionCount
- CK_ULONG ulRwSessionCount
- CK_ULONG ulMaxPinLen
- CK_ULONG ulMinPinLen
- CK_ULONG ulTotalPublicMemory
- CK_ULONG ulFreePublicMemory
- CK_ULONG ulTotalPrivateMemory
- CK_ULONG ulFreePrivateMemory
- CK_VERSION hardwareVersion
- CK_VERSION firmwareVersion
- CK_CHAR utcTime [16]

9.117.1 Field Documentation

9.117.1.1 firmwareVersion

```
CK_VERSION firmwareVersion
```

9.117.1.2 flags

```
CK_FLAGS flags
```

9.117.1.3 hardwareVersion

```
CK_VERSION hardwareVersion
```

9.117.1.4 label

```
CK_UTF8CHAR label[32]
```

9.117.1.5 manufacturerID

```
CK_UTF8CHAR manufacturerID[32]
```

9.117.1.6 model

```
CK_UTF8CHAR model[16]
```

9.117.1.7 serialNumber

```
CK_CHAR serialNumber[16]
```

9.117.1.8 ulFreePrivateMemory

```
CK_ULONG ulFreePrivateMemory
```

9.117.1.9 ulFreePublicMemory

```
CK_ULONG ulFreePublicMemory
```

9.117.1.10 ulMaxPinLen

```
CK_ULONG ulMaxPinLen
```

9.117.1.11 ulMaxRwSessionCount

```
CK_ULONG ulMaxRwSessionCount
```

9.117.1.12 ulMaxSessionCount

```
CK_ULONG ulMaxSessionCount
```

9.117.1.13 ulMinPinLen

```
CK_ULONG ulMinPinLen
```

9.117.1.14 ulRwSessionCount

```
CK_ULONG ulRwSessionCount
```

9.117.1.15 ulSessionCount

```
CK_ULONG ulSessionCount
```

9.117.1.16 ulTotalPrivateMemory

```
CK_ULONG ulTotalPrivateMemory
```

9.117.1.17 ulTotalPublicMemory

```
CK_ULONG ulTotalPublicMemory
```

9.117.1.18 utcTime

```
CK_CHAR utcTime[16]
```

9.118 CK_VERSION Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE major
- CK_BYTE minor

9.118.1 Field Documentation

9.118.1.1 major

```
CK_BYTE major
```

9.118.1.2 minor

```
CK_BYTE minor
```

9.119 CK_WTLS_KEY_MAT_OUT Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_OBJECT_HANDLE hMacSecret
- CK_OBJECT_HANDLE hKey
- CK_BYTE_PTR pIV

9.119.1 Field Documentation

9.119.1.1 hKey

```
CK_OBJECT_HANDLE hKey
```

9.119.1.2 hMacSecret

```
CK_OBJECT_HANDLE hMacSecret
```

9.119.1.3 pIV

```
CK_BYTE_PTR pIV
```

9.120 CK_WTLS_KEY_MAT_PARAMS Struct Reference

```
#include &lt;pkcs11t.h&gt;
```

Data Fields

- CK_MECHANISM_TYPE DigestMechanism
- CK_ULONG ulMacSizeInBits
- CK_ULONG ulKeySizeInBits
- CK_ULONG ulIVSizeInBits
- CK_ULONG ulSequenceNumber
- CK_BBOOL bIsExport
- CK_WTLS_RANDOM_DATA RandomInfo
- CK_WTLS_KEY_MAT_OUT_PTR pReturnedKeyMaterial

9.120.1 Field Documentation

9.120.1.1 bIsExport

```
CK_BBOOL bIsExport
```

9.120.1.2 DigestMechanism

```
CK_MECHANISM_TYPE DigestMechanism
```

9.120.1.3 pReturnedKeyMaterial

```
CK_WTLS_KEY_MAT_OUT_PTR pReturnedKeyMaterial
```

9.120.1.4 RandomInfo

```
CK_WTLS_RANDOM_DATA RandomInfo
```

9.120.1.5 ulIVSizeInBits

```
CK_ULONG ulIVSizeInBits
```

9.120.1.6 ulKeySizeInBits

```
CK_ULONG ulKeySizeInBits
```

9.120.1.7 ulMacSizeInBits

```
CK_ULONG ulMacSizeInBits
```

9.120.1.8 ulSequenceNumber

```
CK_ULONG ulSequenceNumber
```

9.121 CK_WTLS_MASTER_KEY_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE DigestMechanism
- CK_WTLS_RANDOM_DATA RandomInfo
- CK_BYTE_PTR pVersion

9.121.1 Field Documentation

9.121.1.1 DigestMechanism

```
CK_MECHANISM_TYPE DigestMechanism
```

9.121.1.2 pVersion

```
CK_BYTE_PTR pVersion
```

9.121.1.3 RandomInfo

```
CK_WTLS_RANDOM_DATA RandomInfo
```

9.122 CK_WTLS_PRF_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_MECHANISM_TYPE DigestMechanism
- CK_BYTE_PTR pSeed
- CK_ULONG ulSeedLen
- CK_BYTE_PTR pLabel
- CK_ULONG ulLabelLen
- CK_BYTE_PTR pOutput
- CK_ULONG_PTR pulOutputLen

9.122.1 Field Documentation

9.122.1.1 DigestMechanism

```
CK_MECHANISM_TYPE DigestMechanism
```

9.122.1.2 pLabel

```
CK_BYTE_PTR pLabel
```

9.122.1.3 pOutput

```
CK_BYTE_PTR pOutput
```

9.122.1.4 pSeed

```
CK_BYTE_PTR pSeed
```

9.122.1.5 pulOutputLen

```
CK_ULONG_PTR pulOutputLen
```

9.122.1.6 ulLabelLen

```
CK_ULONG ulLabelLen
```

9.122.1.7 ulSeedLen

```
CK_ULONG ulSeedLen
```

9.123 CK_WTLS_RANDOM_DATA Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_BYTE_PTR pClientRandom
- CK_ULONG ulClientRandomLen
- CK_BYTE_PTR pServerRandom
- CK_ULONG ulServerRandomLen

9.123.1 Field Documentation

9.123.1.1 pClientRandom

```
CK_BYTE_PTR pClientRandom
```

9.123.1.2 pServerRandom

```
CK_BYTE_PTR pServerRandom
```

9.123.1.3 ulClientRandomLen

```
CK_ULONG ulClientRandomLen
```

9.123.1.4 ulServerRandomLen

```
CK_ULONG ulServerRandomLen
```

9.124 CK_X9_42_DH1_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_X9_42_DH_KDF_TYPE kdf
- CK_ULONG ulOtherInfoLen
- CK_BYTE_PTR pOtherInfo
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData

9.124.1 Field Documentation

9.124.1.1 kdf

```
CK_X9_42_DH_KDF_TYPE kdf
```

9.124.1.2 pOtherInfo

```
CK_BYTE_PTR pOtherInfo
```

9.124.1.3 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.124.1.4 ulOtherInfoLen

```
CK_ULONG ulOtherInfoLen
```

9.124.1.5 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.125 CK_X9_42_DH2_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_X9_42_DH_KDF_TYPE kdf
- CK_ULONG ulOtherInfoLen
- CK_BYTE_PTR pOtherInfo
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPrivateDataLen
- CK_OBJECT_HANDLE hPrivateData
- CK_ULONG ulPublicDataLen2
- CK_BYTE_PTR pPublicData2

9.125.1 Field Documentation

9.125.1.1 hPrivateData

```
CK_OBJECT_HANDLE hPrivateData
```

9.125.1.2 kdf

```
CK_X9_42_DH_KDF_TYPE kdf
```

9.125.1.3 pOtherInfo

```
CK_BYTE_PTR pOtherInfo
```

9.125.1.4 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.125.1.5 pPublicData2

```
CK_BYTE_PTR pPublicData2
```

9.125.1.6 ulOtherInfoLen

```
CK_ULONG ulOtherInfoLen
```

9.125.1.7 ulPrivateDataLen

```
CK_ULONG ulPrivateDataLen
```

9.125.1.8 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.125.1.9 ulPublicDataLen2

```
CK_ULONG ulPublicDataLen2
```

9.126 CK_X9_42_MQV_DERIVE_PARAMS Struct Reference

```
#include <pkcs11t.h>
```

Data Fields

- CK_X9_42_DH_KDF_TYPE kdf
- CK_ULONG ulOtherInfoLen
- CK_BYTE_PTR pOtherInfo
- CK_ULONG ulPublicDataLen
- CK_BYTE_PTR pPublicData
- CK_ULONG ulPrivateDataLen
- CK_OBJECT_HANDLE hPrivateData
- CK_ULONG ulPublicDataLen2
- CK_BYTE_PTR pPublicData2
- CK_OBJECT_HANDLE publicKey

9.126.1 Field Documentation

9.126.1.1 hPrivateData

```
CK_OBJECT_HANDLE hPrivateData
```

9.126.1.2 kdf

```
CK_X9_42_DH_KDF_TYPE kdf
```

9.126.1.3 pOtherInfo

```
CK_BYTE_PTR pOtherInfo
```

9.126.1.4 pPublicData

```
CK_BYTE_PTR pPublicData
```

9.126.1.5 pPublicData2

```
CK_BYTE_PTR pPublicData2
```

9.126.1.6 publicKey

```
CK_OBJECT_HANDLE publicKey
```

9.126.1.7 ulOtherInfoLen

```
CK_ULONG ulOtherInfoLen
```

9.126.1.8 ulPrivateDataLen

```
CK_ULONG ulPrivateDataLen
```

9.126.1.9 ulPublicDataLen

```
CK_ULONG ulPublicDataLen
```

9.126.1.10 ulPublicDataLen2

```
CK_ULONG ulPublicDataLen2
```

9.127 CL_HashContext Struct Reference

```
#include <sha1_routines.h>
```

Data Fields

- uint32_t h [20/4]
- uint32_t buf [64/4]
- uint32_t byteCount
- uint32_t byteCountHi

9.127.1 Field Documentation

9.127.1.1 buf

```
uint32_t buf[64/4]
```

9.127.1.2 byteCount

```
uint32_t byteCount
```

9.127.1.3 byteCountHi

```
uint32_t byteCountHi
```

9.127.1.4 h

```
uint32_t h[20/4]
```

9.128 hw_sha256_ctx Struct Reference

Data Fields

- uint32_t total_msg_size

  Total number of message bytes processed.
- uint32_t block_size

  Number of bytes in current block.
- uint8_t block [ATCA_SHA256_BLOCK_SIZE *2]

  Unprocessed message storage.

9.128.1 Field Documentation

9.128.1.1 block

```
uint8_t block[ATCA_SHA256_BLOCK_SIZE *2]
```

Unprocessed message storage.

9.128.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

9.128.1.3 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

9.129 i2c_sam0_instance Struct Reference

```
#include <hal_sam0_i2c_asf.h>
```

Data Fields

- struct i2c_master_module * i2c_instance
- sam0_change_baudrate change_baudrate

9.129.1 Field Documentation

9.129.1.1 change_baudrate

```
sam0_change_baudrate change_baudrate
```

9.129.1.2 i2c_instance

```
struct i2c_master_module* i2c_instance
```

9.130 i2c_sam_instance Struct Reference

```
#include <hal_sam_i2c_asf.h>
```

Data Fields

- Twi ∗ i2c_instance
- sam_change_baudrate change_baudrate

9.130.1 Field Documentation

9.130.1.1 change_baudrate

```
sam_change_baudrate change_baudrate
```

9.130.1.2 i2c_instance

```
Twi* i2c_instance
```

9.131 i2c_start_instance Struct Reference

```
#include <hal_i2c_start.h>
```

Data Fields

- struct i2c_m_sync_desc ∗ i2c_descriptor
- start_change_baudrate change_baudrate

9.131.1 Field Documentation

9.131.1.1 change_baudrate

```
start_change_baudrate change_baudrate
```

9.131.1.2 i2c_descriptor

```
struct i2c_m_sync_desc* i2c_descriptor
```

9.132 memory_parameters Struct Reference

```
#include <secure_boot_memory.h>
```

Data Fields

- uint32_t start_address
- uint32_t memory_size
- uint32_t version_info
- uint8_t reserved [52]
- uint8_t signature [ATCA_SIG_SIZE]

9.132.1 Field Documentation

9.132.1.1 memory_size

```
uint32_t memory_size
```

9.132.1.2 reserved

```
uint8_t reserved[52]
```

9.132.1.3 signature

```
uint8_t signature[ATCA_SIG_SIZE]
```

9.132.1.4 start_address

```
uint32_t start_address
```

9.132.1.5 version_info

```
uint32_t version_info
```

9.133 secure_boot_config_bits Struct Reference

```
#include <secure_boot.h>
```

Data Fields

- uint16_t secure_boot_mode: 2
- uint16_t secure_boot_reserved1: 1
- uint16_t secure_boot_persistent_enable: 1
- uint16_t secure_boot_rand_nonce: 1
- uint16_t secure_boot_reserved2: 3
- uint16_t secure_boot_sig_dig: 4
- uint16_t secure_boot_pub_key: 4

9.133.1 Field Documentation

9.133.1.1 secure_boot_mode

```
uint16_t secure_boot_mode
```

9.133.1.2 secure_boot_persistent_enable

```
uint16_t secure_boot_persistent_enable
```

9.133.1.3 secure_boot_pub_key

```
uint16_t secure_boot_pub_key
```

9.133.1.4 secure_boot_rand_nonce

```
uint16_t secure_boot_rand_nonce
```

9.133.1.5 secure_boot_reserved1

```
uint16_t secure_boot_reserved1
```

9.133.1.6 secure_boot_reserved2

```
uint16_t secure_boot_reserved2
```

9.133.1.7 secure_boot_sig_dig

```
uint16_t secure_boot_sig_dig
```

9.134 secure_boot_parameters Struct Reference

```
#include <secure_boot.h>
```

Data Fields

- memory_parameters memory_params
- atcac_sha2_256_ctx s_sha_context
- uint8_t app_digest [ATCA_SHA_DIGEST_SIZE]

9.134.1 Field Documentation

9.134.1.1 app_digest

```
uint8_t app_digest[ATCA_SHA_DIGEST_SIZE]
```

9.134.1.2 memory_params

```
memory_parameters memory_params
```

9.134.1.3 s_sha_context

```
atcac_sha2_256_ctx s_sha_context
```

9.135 sw_sha256_ctx Struct Reference

```
#include <sha2_routines.h>
```

Data Fields

- uint32_t total_msg_size
  Total number of message bytes processed.
- uint32_t block_size
  Number of bytes in current block.
- uint8_t block [(64) *2]
  Unprocessed message storage.
- uint32_t hash [8]
  Hash state.

9.135.1 Field Documentation

9.135.1.1 block

```
uint8_t block[(64) *2]
```

Unprocessed message storage.

9.135.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

9.135.1.3 hash

```
uint32_t hash[8]
```

Hash state.

9.135.1.4 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

9.136 tng_cert_map_element Struct Reference

Data Fields

- const char * otpcode
- const atcacert_def_t * cert_def

9.136.1 Field Documentation

9.136.1.1 cert_def

```
const atcacert_def_t* cert_def
```

9.136.1.2 otpcode

```
const char* otpcode
```

Chapter 10

File Documentation

10.1 api_206a.c File Reference

Provides APIs to use with ATSHA206A device.

```
#include <stdlib.h>
#include <stdio.h>
#include "cryptoauthlib.h"
#include "api_206a.h"
```

Functions

- ATCA_STATUS sha206a_diversify_parent_key (uint8_t *parent_key, uint8_t *diversified_key)

  Computes the diversified key based on the parent key provided and device serial number.
- ATCA_STATUS sha206a_generate_derive_key (uint8_t *parent_key, uint8_t *derived_key, uint8_t param1, uint16_t param2)

  Generates the derived key based on the parent key and other parameters provided.
- ATCA_STATUS sha206a_generate_challenge_response_pair (uint8_t *key, uint8_t *challenge, uint8_↩ t *response)

  Generates the response based on Key and Challenge provided.
- ATCA_STATUS sha206a_authenticate (uint8_t *challenge, uint8_t *expected_response, uint8_t *is_↩ authenticated)

  verifies the challenge and provided response using key in device
- ATCA_STATUS sha206a_verify_device_consumption (uint8_t *is_consumed)

  verifies the device is fully consumed or not based on Parent and Derived Key use flags.
- ATCA_STATUS sha206a_check_dk_useflag_validity (uint8_t *is_consumed)

  verifies Derived Key use flags for consumption
- ATCA_STATUS sha206a_check_pk_useflag_validity (uint8_t *is_consumed)

  verifies Parent Key use flags for consumption
- ATCA_STATUS sha206a_get_dk_useflag_count (uint8_t *dk_available_count)

  calculates available Derived Key use counts
- ATCA_STATUS sha206a_get_pk_useflag_count (uint8_t *pk_available_count)

  calculates available Parent Key use counts
- ATCA_STATUS sha206a_get_dk_update_count (uint8_t *dk_update_count)

  Read Derived Key slot update count. It will be wraps around 256.

- ATCA_STATUS sha206a_write_data_store (uint8_t slot, uint8_t *data, uint8_t block, uint8_t offset, uint8_t len, bool lock_after_write)

  Update the data store slot with user data and lock it if necessary.
- ATCA_STATUS sha206a_read_data_store (uint8_t slot, uint8_t *data, uint8_t offset, uint8_t len)

  Read the data stored in Data store.
- ATCA_STATUS sha206a_get_data_store_lock_status (uint8_t slot, uint8_t *is_locked)

  Returns the lock status of the given data store.

10.1.1 Detailed Description

Provides APIs to use with ATSHA206A device.

Copyright

10.1.2 Function Documentation

10.1.2.1 sha206a_authenticate()

```
ATCA_STATUS sha206a_authenticate (
            uint8_t * challenge,
            uint8_t * expected_response,
            uint8_t * is_authenticated )
```

verifies the challenge and provided response using key in device

Parameters

in	*challenge*	Challenge to be used in the response calculations
in	*expected_response*	Expected response from the device.
out	*is_authenticated*	result of expected of response and calcualted response

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.2 sha206a_check_dk_useflag_validity()

```
ATCA_STATUS sha206a_check_dk_useflag_validity (
            uint8_t * is_consumed )
```

verifies Derived Key use flags for consumption

Parameters

out	*is_consumed*	indicates if DK is available for consumption.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.3 sha206a_check_pk_useflag_validity()

```
ATCA_STATUS sha206a_check_pk_useflag_validity (
            uint8_t * is_consumed )
```

verifies Parent Key use flags for consumption

Parameters

out	*is_consumed*	indicates if PK is available for consumption

Returns

ATCA_SUCCESS on success, otherwise an error code

10.1.2.4 sha206a_diversify_parent_key()

```
ATCA_STATUS sha206a_diversify_parent_key (
            uint8_t * parent_key,
            uint8_t * diversified_key )
```

Computes the diversified key based on the parent key provided and device serial number.

Parameters

in	*parent_key*	parent key to be diversified
out	*diversified_key*	diversified parent key

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.5 sha206a_generate_challenge_response_pair()

```
ATCA_STATUS sha206a_generate_challenge_response_pair (
            uint8_t * key,
            uint8_t * challenge,
            uint8_t * response )
```

Generates the response based on Key and Challenge provided.

Parameters

in	*key*	Input data contains device's key
in	*challenge*	Input data to be used in challenge response calculation
out	*response*	response derived from key and challenge

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.6 sha206a_generate_derive_key()

```
ATCA_STATUS sha206a_generate_derive_key (
            uint8_t * parent_key,
            uint8_t * derived_key,
            uint8_t param1,
            uint16_t param2 )
```

Generates the derived key based on the parent key and other parameters provided.

Parameters

in	*parent_key*	Input data contains device's parent key
out	*derived_key*	Output data derived from parent key
in	*param1*	Input data to be used in derive key calculation
in	*param2*	Input data to be used in derive key calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.7 sha206a_get_data_store_lock_status()

```
ATCA_STATUS sha206a_get_data_store_lock_status (
            uint8_t slot,
            uint8_t * is_locked )
```

Returns the lock status of the given data store.

Parameters

in	*slot*	Slot number of the data store
out	*is_locked*	lock status of the data store

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.8 sha206a_get_dk_update_count()

```
ATCA_STATUS sha206a_get_dk_update_count (
            uint8_t * dk_update_count )
```

Read Derived Key slot update count. It will be wraps around 256.

Parameters

out	*dk_update_count*	returns number of times the slot has been updated with derived key

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.9 sha206a_get_dk_useflag_count()

```
ATCA_STATUS sha206a_get_dk_useflag_count (
            uint8_t * dk_available_count )
```

calculates available Derived Key use counts

Parameters

out	*dk_available_count*	counts available bit's as 1

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.10 sha206a_get_pk_useflag_count()

```
ATCA_STATUS sha206a_get_pk_useflag_count (
            uint8_t * pk_available_count )
```

calculates available Parent Key use counts

Parameters

out	*pk_available_count*	counts available bit's as 1

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.11 sha206a_read_data_store()

```
ATCA_STATUS sha206a_read_data_store (
            uint8_t slot,
            uint8_t * data,
            uint8_t offset,
            uint8_t len )
```

Read the data stored in Data store.

Parameters

in	*slot*	Slot number to read from
in	*data*	Pointer to hold slot data data
in	*offset*	Byte offset within the zone to read from.
in	*len*	data length

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.12 sha206a_verify_device_consumption()

```
ATCA_STATUS sha206a_verify_device_consumption (
            uint8_t * is_consumed )
```

verifies the device is fully consumed or not based on Parent and Derived Key use flags.

Parameters

out	*is_consumed*	result of device consumption

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.1.2.13 sha206a_write_data_store()

```
ATCA_STATUS sha206a_write_data_store (
            uint8_t slot,
            uint8_t * data,
            uint8_t block,
            uint8_t offset,
            uint8_t len,
            bool lock_after_write )
```

Update the data store slot with user data and lock it if necessary.

Parameters

in	*slot*	Slot number to be written with data
in	*data*	Pointer that holds the data
in	*block*	32-byte block to write to.
in	*offset*	4-byte word within the specified block to write to. If performing a 32-byte write, this should be 0.
in	*len*	data length
in	*lock_after_write*	set 1 to lock slot after write, otherwise 0

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2 api_206a.h File Reference

Provides api interfaces to use with ATSHA206A device.

```
#include "atca_status.h"
```

Macros

- #define ATCA_SHA206A_ZONE_WRITE_LOCK 0x20
- #define ATCA_SHA206A_DKEY_CONSUMPTION_MASK 0x01
- #define ATCA_SHA206A_PKEY_CONSUMPTION_MASK 0x02
- #define ATCA_SHA206A_SYMMETRIC_KEY_ID_SLOT 0X07

Enumerations

- enum { SHA206A_DATA_STORE0 =8, SHA206A_DATA_STORE1, SHA206A_DATA_STORE2 }

Functions

- ATCA_STATUS sha206a_diversify_parent_key (uint8_t *parent_key, uint8_t *diversified_key)

  Computes the diversified key based on the parent key provided and device serial number.
- ATCA_STATUS sha206a_generate_derive_key (uint8_t *parent_key, uint8_t *derived_key, uint8_t param1, uint16_t param2)

  Generates the derived key based on the parent key and other parameters provided.
- ATCA_STATUS sha206a_generate_challenge_response_pair (uint8_t *key, uint8_t *challenge, uint8_↩ t *response)

  Generates the response based on Key and Challenge provided.
- ATCA_STATUS sha206a_authenticate (uint8_t *challenge, uint8_t *expected_response, uint8_t *is_↩ authenticated)

  verifies the challenge and provided response using key in device
- ATCA_STATUS sha206a_verify_device_consumption (uint8_t *is_consumed)

  verifies the device is fully consumed or not based on Parent and Derived Key use flags.
- ATCA_STATUS sha206a_check_dk_useflag_validity (uint8_t *is_valid)

  verifies Derived Key use flags for consumption
- ATCA_STATUS sha206a_check_pk_useflag_validity (uint8_t *is_valid)

  verifies Parent Key use flags for consumption
- ATCA_STATUS sha206a_get_dk_useflag_count (uint8_t *dk_available_count)

  calculates available Derived Key use counts
- ATCA_STATUS sha206a_get_pk_useflag_count (uint8_t *pk_available_count)

  calculates available Parent Key use counts
- ATCA_STATUS sha206a_get_dk_update_count (uint8_t *dk_update_count)

  Read Derived Key slot update count. It will be wraps around 256.
- ATCA_STATUS sha206a_write_data_store (uint8_t slot, uint8_t *data, uint8_t block, uint8_t offset, uint8_t len, bool lock_after_write)

  Update the data store slot with user data and lock it if necessary.
- ATCA_STATUS sha206a_read_data_store (uint8_t slot, uint8_t *data, uint8_t offset, uint8_t len)

  Read the data stored in Data store.
- ATCA_STATUS sha206a_get_data_store_lock_status (uint8_t slot, uint8_t *is_locked)

  Returns the lock status of the given data store.

10.2.1 Detailed Description

Provides api interfaces to use with ATSHA206A device.

Copyright

10.2.2 Macro Definition Documentation

10.2.2.1 ATCA_SHA206A_DKEY_CONSUMPTION_MASK

```
#define ATCA_SHA206A_DKEY_CONSUMPTION_MASK 0x01
```

10.2.2.2 ATCA_SHA206A_PKEY_CONSUMPTION_MASK

```
#define ATCA_SHA206A_PKEY_CONSUMPTION_MASK 0x02
```

10.2.2.3 ATCA_SHA206A_SYMMETRIC_KEY_ID_SLOT

```
#define ATCA_SHA206A_SYMMETRIC_KEY_ID_SLOT 0X07
```

10.2.2.4 ATCA_SHA206A_ZONE_WRITE_LOCK

```
#define ATCA_SHA206A_ZONE_WRITE_LOCK 0x20
```

10.2.3 Enumeration Type Documentation

10.2.3.1 anonymous enum

```
anonymous enum
```

Enumerator

SHA206A_DATA_STORE0	
SHA206A_DATA_STORE1	
SHA206A_DATA_STORE2	

10.2.4 Function Documentation

10.2.4.1 sha206a_authenticate()

```
ATCA_STATUS sha206a_authenticate (
            uint8_t * challenge,
            uint8_t * expected_response,
            uint8_t * is_authenticated )
```

verifies the challenge and provided response using key in device

Parameters

in	*challenge*	Challenge to be used in the response calculations
in	*expected_response*	Expected response from the device.
out	*is_authenticated*	result of expected of response and calcualted response

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.2 sha206a_check_dk_useflag_validity()

```
ATCA_STATUS sha206a_check_dk_useflag_validity (
            uint8_t * is_consumed )
```

verifies Derived Key use flags for consumption

Parameters

out	*is_consumed*	indicates if DK is available for consumption.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.3 sha206a_check_pk_useflag_validity()

```
ATCA_STATUS sha206a_check_pk_useflag_validity (
            uint8_t * is_consumed )
```

verifies Parent Key use flags for consumption

Parameters

out	*is_consumed*	indicates if PK is available for consumption

Returns

ATCA_SUCCESS on success, otherwise an error code

10.2.4.4 sha206a_diversify_parent_key()

```
ATCA_STATUS sha206a_diversify_parent_key (
            uint8_t * parent_key,
            uint8_t * diversified_key )
```

Computes the diversified key based on the parent key provided and device serial number.

Parameters

in	*parent_key*	parent key to be diversified
out	*diversified_key*	diversified parent key

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.5 sha206a_generate_challenge_response_pair()

```
ATCA_STATUS sha206a_generate_challenge_response_pair (
            uint8_t * key,
            uint8_t * challenge,
            uint8_t * response )
```

Generates the response based on Key and Challenge provided.

Parameters

in	*key*	Input data contains device's key
in	*challenge*	Input data to be used in challenge response calculation
out	*response*	response derived from key and challenge

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.6 sha206a_generate_derive_key()

```
ATCA_STATUS sha206a_generate_derive_key (
            uint8_t * parent_key,
            uint8_t * derived_key,
            uint8_t param1,
            uint16_t param2 )
```

Generates the derived key based on the parent key and other parameters provided.

Parameters

in	*parent_key*	Input data contains device's parent key
out	*derived_key*	Output data derived from parent key
in	*param1*	Input data to be used in derive key calculation
in	*param2*	Input data to be used in derive key calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.7 sha206a_get_data_store_lock_status()

```
ATCA_STATUS sha206a_get_data_store_lock_status (
            uint8_t slot,
            uint8_t * is_locked )
```

Returns the lock status of the given data store.

Parameters

in	*slot*	Slot number of the data store
out	*is_locked*	lock status of the data store

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.8 sha206a_get_dk_update_count()

```
ATCA_STATUS sha206a_get_dk_update_count (
            uint8_t * dk_update_count )
```

Read Derived Key slot update count. It will be wraps around 256.

Parameters

out	*dk_update_count*	returns number of times the slot has been updated with derived key

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.9 sha206a_get_dk_useflag_count()

```
ATCA_STATUS sha206a_get_dk_useflag_count (
            uint8_t * dk_available_count )
```

calculates available Derived Key use counts

Parameters

out	*dk_available_count*	counts available bit's as 1

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.10 sha206a_get_pk_useflag_count()

```
ATCA_STATUS sha206a_get_pk_useflag_count (
            uint8_t * pk_available_count )
```

calculates available Parent Key use counts

Parameters

out	*pk_available_count*	counts available bit's as 1

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.11 sha206a_read_data_store()

```
ATCA_STATUS sha206a_read_data_store (
            uint8_t slot,
            uint8_t * data,
            uint8_t offset,
            uint8_t len )
```

Read the data stored in Data store.

Parameters

in	*slot*	Slot number to read from
in	*data*	Pointer to hold slot data data
in	*offset*	Byte offset within the zone to read from.
in	*len*	data length

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.12 sha206a_verify_device_consumption()

```
ATCA_STATUS sha206a_verify_device_consumption (
            uint8_t * is_consumed )
```

verifies the device is fully consumed or not based on Parent and Derived Key use flags.

Parameters

out	*is_consumed*	result of device consumption

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.2.4.13 sha206a_write_data_store()

```
ATCA_STATUS sha206a_write_data_store (
            uint8_t slot,
            uint8_t * data,
            uint8_t block,
            uint8_t offset,
            uint8_t len,
            bool lock_after_write )
```

Update the data store slot with user data and lock it if necessary.

Parameters

in	*slot*	Slot number to be written with data
in	*data*	Pointer that holds the data
in	*block*	32-byte block to write to.
in	*offset*	4-byte word within the specified block to write to. If performing a 32-byte write, this should be 0.
in	*len*	data length
in	*lock_after_write*	set 1 to lock slot after write, otherwise 0

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.3 ascii_kit_host.c File Reference

KIT protocol intepreter.

```
#include <ctype.h>
#include "ascii_kit_host.h"
#include "hal/kit_protocol.h"
#include "talib/talib_fce.h"
```

Functions

- ATCA_STATUS kit_host_init_phy (atca_hal_kit_phy_t *phy, ATCAIface iface)

  Initializes a phy structure with a cryptoauthlib hal adapter.
- ATCA_STATUS kit_host_init (ascii_kit_host_context_t *ctx, ATCAIfaceCfg *iface[ ], const size_t iface_count, const atca_hal_kit_phy_t *phy, const uint32_t flags)

  Initializes the kit protocol parser context.
- size_t kit_host_format_response (uint8_t *response, size_t rlen, ATCA_STATUS status, uint8_t *data, size_t dlen)

  Format the status and data into the kit protocol response format.
- ATCA_STATUS kit_host_process_cmd (ascii_kit_host_context_t *ctx, const kit_host_map_entry_t *cmd_list, int argc, char *argv[ ], uint8_t *response, size_t *rlen)

  Iterate through a command list to match the given command and then will execute it.
- ATCA_STATUS kit_host_process_ta (ascii_kit_host_context_t *ctx, int argc, char *argv[ ], uint8_t *response, size_t *rlen)
- ATCA_STATUS kit_host_process_line (ascii_kit_host_context_t *ctx, uint8_t *input_line, size_t ilen, uint8_t *response, size_t *rlen)

  Parse a line as a kit protocol command. The kit protocol is printable ascii and each line ends with a newline character.
- void kit_host_task (ascii_kit_host_context_t *ctx)

  Non returning kit protocol runner using the configured physical interface that was provided when the context was initialized.

10.3.1 Detailed Description

KIT protocol intepreter.

Copyright

(c) 2018 Microchip Technology Inc. and its subsidiaries. You may use this software and any derivatives exclusively with Microchip products.

10.3.2 Function Documentation

10.3.2.1 kit_host_format_response()

```
size_t kit_host_format_response (
            uint8_t * response,
            size_t rlen,
            ATCA_STATUS status,
            uint8_t * data,
            size_t dlen )
```

Format the status and data into the kit protocol response format.

10.3.2.2 kit_host_init()

```
ATCA_STATUS kit_host_init (
            ascii_kit_host_context_t * ctx,
            ATCAIfaceCfg * iface[ ],
            const size_t iface_count,
            const atca_hal_kit_phy_t * phy,
            const uint32_t flags )
```

Initializes the kit protocol parser context.

Returns

ATCA_SUCCESS on success, otherwise an error code

Parameters

ctx	Kit protocol parser context
iface	List of device configurations which will be used
iface_count	Number of configurations provided
phy	Kit protocol physical adapter
flags	Option Flags

10.3.2.3 kit_host_init_phy()

```
ATCA_STATUS kit_host_init_phy (
            atca_hal_kit_phy_t * phy,
            ATCAIface iface )
```

Initializes a phy structure with a cryptoauthlib hal adapter.

Returns

ATCA_SUCCESS on success, otherwise an error code

10.3.2.4 kit_host_process_cmd()

```
ATCA_STATUS kit_host_process_cmd (
            ascii_kit_host_context_t * ctx,
            const kit_host_map_entry_t * cmd_list,
            int argc,
            char * argv[],
            uint8_t * response,
            size_t * rlen )
```

Iterate through a command list to match the given command and then will execute it.

10.3.2.5 kit_host_process_line()

```
ATCA_STATUS kit_host_process_line (
            ascii_kit_host_context_t * ctx,
            uint8_t * input_line,
            size_t ilen,
            uint8_t * response,
            size_t * rlen )
```

Parse a line as a kit protocol command. The kit protocol is printable ascii and each line ends with a newline character.

10.3.2.6 kit_host_process_ta()

```
ATCA_STATUS kit_host_process_ta (
            ascii_kit_host_context_t * ctx,
            int argc,
            char * argv[],
            uint8_t * response,
            size_t * rlen )
```

10.3.2.7 kit_host_task()

```
void kit_host_task (
            ascii_kit_host_context_t * ctx )
```

Non returning kit protocol runner using the configured physical interface that was provided when the context was initialized.

10.4 ascii_kit_host.h File Reference

KIT protocol intepreter.

```
#include "cryptoauthlib.h"
```

Data Structures

- struct _ascii_kit_host_context
- struct _kit_host_map_entry

Macros

- #define KIT_LAYER_DELIMITER ':'
- #define KIT_DATA_BEGIN_DELIMITER '('
- #define KIT_DATA_END_DELIMITER ')'
- #define KIT_MESSAGE_DELIMITER '\n'
- #define KIT_MESSAGE_SIZE_MAX (2500)

  The Kit Protocol maximum message size.
- #define KIT_SECTION_NAME_SIZE_MAX KIT_MESSAGE_SIZE_MAX
- #define KIT_VERSION_SIZE_MAX (32)
- #define KIT_FIRMWARE_SIZE_MAX (32)

Typedefs

- typedef struct _ascii_kit_host_context ascii_kit_host_context_t
- typedef struct _kit_host_map_entry kit_host_map_entry_t

Functions

- ATCA_STATUS kit_host_init_phy (atca_hal_kit_phy_t ∗phy, ATCAIface iface)

  Initializes a phy structure with a cryptoauthlib hal adapter.
- ATCA_STATUS kit_host_init (ascii_kit_host_context_t ∗ctx, ATCAIfaceCfg ∗iface[ ], const size_t iface_count, const atca_hal_kit_phy_t ∗phy, const uint32_t flags)

  Initializes the kit protocol parser context.
- size_t kit_host_format_response (uint8_t ∗response, size_t rlen, ATCA_STATUS status, uint8_t ∗data, size_t dlen)

  Format the status and data into the kit protocol response format.
- ATCA_STATUS kit_host_process_cmd (ascii_kit_host_context_t ∗ctx, const kit_host_map_entry_t ∗cmd_list, int argc, char ∗argv[ ], uint8_t ∗response, size_t ∗rlen)

  Iterate through a command list to match the given command and then will execute it.
- ATCA_STATUS kit_host_process_line (ascii_kit_host_context_t ∗ctx, uint8_t ∗input_line, size_t ilen, uint8_t ∗response, size_t ∗rlen)

  Parse a line as a kit protocol command. The kit protocol is printable ascii and each line ends with a newline character.
- void kit_host_task (ascii_kit_host_context_t ∗ctx)

  Non returning kit protocol runner using the configured physical interface that was provided when the context was initialized.

10.4.1 Detailed Description

KIT protocol intepreter.

Copyright

10.4.2 Macro Definition Documentation

10.4.2.1 KIT_DATA_BEGIN_DELIMITER

```
#define KIT_DATA_BEGIN_DELIMITER '('
```

10.4.2.2 KIT_DATA_END_DELIMITER

```
#define KIT_DATA_END_DELIMITER ')'
```

10.4.2.3 KIT_FIRMWARE_SIZE_MAX

```
#define KIT_FIRMWARE_SIZE_MAX (32)
```

10.4.2.4 KIT_LAYER_DELIMITER

```
#define KIT_LAYER_DELIMITER ':'
```

10.4.2.5 KIT_MESSAGE_DELIMITER

```
#define KIT_MESSAGE_DELIMITER '\n'
```

10.4.2.6 KIT_MESSAGE_SIZE_MAX

```
#define KIT_MESSAGE_SIZE_MAX (2500)
```

The Kit Protocol maximum message size.

Note

Send: <target>:<command>(optional hex bytes to send)
Receive: <status hex byte>(optional hex bytes of response)

10.4.2.7 KIT_SECTION_NAME_SIZE_MAX

```
#define KIT_SECTION_NAME_SIZE_MAX KIT_MESSAGE_SIZE_MAX
```

10.4.2.8 KIT_VERSION_SIZE_MAX

```
#define KIT_VERSION_SIZE_MAX (32)
```

10.4.3 Typedef Documentation

10.4.3.1 ascii_kit_host_context_t

```
typedef struct _ascii_kit_host_context ascii_kit_host_context_t
```

10.4.3.2 kit_host_map_entry_t

```
typedef struct _kit_host_map_entry kit_host_map_entry_t
```

Used to create command tables for the kit host parser

10.4.4 Function Documentation

10.4.4.1 kit_host_format_response()

```
size_t kit_host_format_response (
            uint8_t * response,
            size_t rlen,
            ATCA_STATUS status,
            uint8_t * data,
            size_t dlen )
```

Format the status and data into the kit protocol response format.

10.4.4.2 kit_host_init()

```
ATCA_STATUS kit_host_init (
            ascii_kit_host_context_t * ctx,
            ATCAIfaceCfg * iface[],
            const size_t iface_count,
            const atca_hal_kit_phy_t * phy,
            const uint32_t flags )
```

Initializes the kit protocol parser context.

Returns

ATCA_SUCCESS on success, otherwise an error code

Parameters

ctx	Kit protocol parser context
iface	List of device configurations which will be used
iface_count	Number of configurations provided
phy	Kit protocol physical adapter
flags	Option Flags

10.4.4.3 kit_host_init_phy()

```
ATCA_STATUS kit_host_init_phy (
            atca_hal_kit_phy_t * phy,
            ATCAIface iface )
```

Initializes a phy structure with a cryptoauthlib hal adapter.

Returns

ATCA_SUCCESS on success, otherwise an error code

10.4.4.4 kit_host_process_cmd()

```
ATCA_STATUS kit_host_process_cmd (
            ascii_kit_host_context_t * ctx,
            const kit_host_map_entry_t * cmd_list,
            int argc,
            char * argv[ ],
            uint8_t * response,
            size_t * rlen )
```

Iterate through a command list to match the given command and then will execute it.

10.4.4.5 kit_host_process_line()

```
ATCA_STATUS kit_host_process_line (
            ascii_kit_host_context_t * ctx,
            uint8_t * input_line,
            size_t ilen,
            uint8_t * response,
            size_t * rlen )
```

Parse a line as a kit protocol command. The kit protocol is printable ascii and each line ends with a newline character.

10.4.4.6 kit_host_task()

```
void kit_host_task (
        ascii_kit_host_context_t * ctx )
```

Non returning kit protocol runner using the configured physical interface that was provided when the context was initialized.

10.5 atca_basic.c File Reference

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

```
#include "atca_basic.h"
#include "atca_version.h"
```

Functions

- ATCA_STATUS atcab_version (char *ver_str)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS atcab_init_ext (ATCADevice *device, ATCAIfaceCfg *cfg)

  Creates and initializes a ATCADevice context.
- ATCA_STATUS atcab_init (ATCAIfaceCfg *cfg)

  Creates a global ATCADevice object used by Basic API.
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

  Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.
- ATCA_STATUS atcab_release_ext (ATCADevice *device)

  release (free) the an ATCADevice instance.
- ATCA_STATUS atcab_release (void)

  release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.
- ATCADevice atcab_get_device (void)

  Get the global device object.
- ATCADeviceType atcab_get_device_type_ext (ATCADevice device)

  Get the selected device type of rthe device context.
- ATCADeviceType atcab_get_device_type (void)

  Get the current device type configured for the global ATCADevice.
- uint8_t atcab_get_device_address (ATCADevice device)

  Get the current device address based on the configured device and interface.
- bool atcab_is_ca_device (ATCADeviceType dev_type)

  Check whether the device is cryptoauth device.
- bool atcab_is_ta_device (ATCADeviceType dev_type)

  Check whether the device is Trust Anchor device.
- ATCA_STATUS atcab_wakeup (void)

  wakeup the CryptoAuth device
- ATCA_STATUS atcab_idle (void)

  idle the CryptoAuth device
- ATCA_STATUS atcab_sleep (void)

invoke sleep on the CryptoAuth device

- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t ∗size)

  Gets the size of the specified zone in bytes.
- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t ∗aes_in, uint8_t ∗aes_out)

  Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS atcab_aes_encrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_gfm (const uint8_t ∗h, const uint8_t ∗input, uint8_t ∗output)

  Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t ∗free_field, size_t free_field_size, uint8_t ∗iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗aad, uint32_t aad_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t ∗ctx, uint8_t ∗tag, size_t tag_size)

  Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗tag, size_t tag_size, bool ∗is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.
- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, const uint8_t ∗response, const uint8_t ∗other_data)

  Compares a MAC response with input values.
- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t ∗counter_value)

  Compute the Counter functions.
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t ∗counter_value)

  Increments one of the device's monotonic counters.
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t ∗counter_value)

  Read one of the device's monotonic counters.
- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t key_id, const uint8_t ∗mac)

  Executes the DeviceKey command for deriving a new key from a nonce (TempKey) and an existing key.

- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t *public_key, uint8_t *pms, uint8_t *out_nonce)

  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t *public_key, uint8_t *pms, const uint8_t *read_key, uint16_t read_key_id, const uint8_t num_in[(20)])

  ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t *other_data, uint8_t other_data_size)

  Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t *other_data, uint8_t *public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t *public_key)

  Issues GenKey command, which generates a new random private key in slot/handle and returns the public key.
- ATCA_STATUS atcab_get_pubkey_ext (ATCADevice device, uint16_t key_id, uint8_t *public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t *public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t *digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t *out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS atcab_info (uint8_t *revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS atcab_info_set_latch (bool state)

  Use the Info command to set the persistent latch state for an ATECC608 device.
- ATCA_STATUS atcab_info_get_latch (bool *state)

  Use the Info command to get the persistent latch current state for an ATECC608 device.
- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t *message, uint8_t *out_data, uint8_t *out_nonce)

  Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

  The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS atcab_lock_config_zone (void)

  Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

  Lock the config zone with summary CRC.

- ATCA_STATUS atcab_lock_data_zone (void)

  Unconditionally (no CRC required) lock the data zone (slots and OTP). for CryptoAuth devices and lock the setup for Trust Anchor device.
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

  Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

  Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1) (for cryptoauth devices) or Lock an individual handle in shared data element on an Trust Anchor device (for Trust Anchor devices).
- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)

  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[(20)])

  Executes PrivWrite command, to write externally generated ECC private keys into the device.
- ATCA_STATUS atcab_random_ext (ATCADevice device, uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the device.
- ATCA_STATUS atcab_random (uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the device.
- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS atcab_is_config_locked (bool ∗is_locked)

  This function check whether configuration zone is locked or not.
- ATCA_STATUS atcab_is_data_locked (bool ∗is_locked)

  This function check whether data/setup zone is locked or not.
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

  This function check whether slot/handle is locked or not.
- ATCA_STATUS atcab_is_private_ext (ATCADevice device, uint16_t slot, bool ∗is_private)

  Check to see if the key is a private key or not.
- ATCA_STATUS atcab_is_private (uint16_t slot, bool ∗is_private)
- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.
- ATCA_STATUS atcab_read_serial_number (uint8_t ∗serial_number)

This function returns serial number of the device.

- ATCA_STATUS atcab_read_pubkey_ext (ATCADevice device, uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t *sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS atcab_read_config_zone (uint8_t *config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS atcab_cmp_config_zone (uint8_t *config_data, bool *same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t *result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608 chip.
- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t *data_in, uint8_t *data_out, uint16_t *data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS atcab_sha_start (void)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS atcab_sha_update (const uint8_t *message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS atcab_sha_end (uint8_t *digest, uint16_t length, const uint8_t *message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_read_context (uint8_t *context, uint16_t *context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS atcab_sha_write_context (const uint8_t *context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t *message, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t *data, size_t data_size, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t *ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t *ctx, uint8_t *digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t *ctx, uint16_t key_slot)

Executes SHA command to start an HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data↩_size)

  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t ∗ctx, uint8_t ∗digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac_ext (ATCADevice device, const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac (const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t ∗signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.

- ATCA_STATUS atcab_sign_ext (ATCADevice device, uint16_t key_id, const uint8_t ∗msg, uint8_↩t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t ∗msg, uint8_t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t ∗signature)

  Executes Sign command to sign an internally generated message.

- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public↩_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.

- ATCA_STATUS atcab_verify_extern_ext (ATCADevice device, const uint8_t ∗message, const uint8_↩t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

- ATCA_STATUS atcab_verify_extern (const uint8_t ∗message, const uint8_t ∗signature, const uint8_↩t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.

- ATCA_STATUS atcab_verify_extern_mac (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.

- ATCA_STATUS atcab_verify_stored_ext (ATCADevice device, const uint8_t ∗message, const uint8_↩t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩Key for other devices.

- ATCA_STATUS atcab_verify_stored (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩Key for other devices.

- ATCA_STATUS atcab_verify_stored_mac (const uint8_t *message, const uint8_t *signature, uint16_t key↩_id, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t *signature, const uint8_t *other_data, bool *is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t *signature, const uint8_t *other_data, bool *is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t *value, const uint8_t *mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_↩t *data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t *data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t *public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS atcab_write_config_zone (const uint8_t *config_data)

  Executes the Write command, which writes the configuration zone.
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

  Initialize one of the monotonic counters in device with a specific value.

Variables

- const char atca_version [ ] = "20210126"
- ATCADevice _gDevice = NULL

10.5.1 Detailed Description

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

Copyright

10.5.2 Variable Documentation

10.5.2.1 atca_version

```
const char atca_version[] = "20210126"
```

10.6 atca_basic.h File Reference

CryptoAuthLib Basic API methods - a simple crypto authentication API. These methods manage a global ATCA↩ Device object behind the scenes. They also manage the wake/idle state transitions so callers don't need to.

```
#include "cryptoauthlib.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "crypto/atca_crypto_hw_aes.h"
```

Macros

- #define atcab_get_addr(...) calib_get_addr(__VA_ARGS__)
- #define atca_execute_command(...) calib_execute_command(__VA_ARGS__)
- #define SHA_CONTEXT_MAX_SIZE (109)

Functions

- ATCA_STATUS atcab_version (char ∗ver_str)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS atcab_init_ext (ATCADevice ∗device, ATCAIfaceCfg ∗cfg)

  Creates and initializes a ATCADevice context.
- ATCA_STATUS atcab_init (ATCAIfaceCfg ∗cfg)

  Creates a global ATCADevice object used by Basic API.
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

  Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.
- ATCA_STATUS atcab_release_ext (ATCADevice ∗device)

  release (free) the an ATCADevice instance.
- ATCA_STATUS atcab_release (void)

  release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.
- ATCADevice atcab_get_device (void)

  Get the global device object.
- ATCADeviceType atcab_get_device_type_ext (ATCADevice device)

  Get the selected device type of rthe device context.
- ATCADeviceType atcab_get_device_type (void)

  Get the current device type configured for the global ATCADevice.
- uint8_t atcab_get_device_address (ATCADevice device)

  Get the current device address based on the configured device and interface.
- bool atcab_is_ca_device (ATCADeviceType dev_type)

  Check whether the device is cryptoauth device.
- bool atcab_is_ta_device (ATCADeviceType dev_type)

  Check whether the device is Trust Anchor device.

- ATCA_STATUS atcab_aes_cbc_init_ext (ATCADevice device, atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbcmac_init_ext (ATCADevice device, atca_aes_cbcmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize context for AES CBC-MAC operation.
- ATCA_STATUS atcab_aes_cbcmac_init (atca_aes_cbcmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize context for AES CBC-MAC operation.
- ATCA_STATUS atcab_aes_cbcmac_update (atca_aes_cbcmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_size)

  Calculate AES CBC-MAC with key stored within ECC608A device. calib_aes_cbcmac_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbcmac_finish (atca_aes_cbcmac_ctx_t ∗ctx, uint8_t ∗mac, uint32_t mac_size)

  Finish a CBC-MAC operation returning the CBC-MAC value. If the data provided to the calib_aes_cbcmac_update() function has incomplete block this function will return an error code.
- ATCA_STATUS atcab_aes_cmac_init_ext (ATCADevice device, atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the device.
- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the device.
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_size)

  Add data to an initialized CMAC calculation.
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t ∗ctx, uint8_t ∗cmac, uint32_t cmac_size)

  Finish a CMAC operation returning the CMAC value.
- ATCA_STATUS atcab_aes_ctr_init_ext (ATCADevice device, atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t ∗iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t ∗iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand_ext (ATCADevice device, atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗input, uint8_t ∗output)

  Process a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t ∗ctx)

  Increments AES CTR counter value.
- ATCA_STATUS atcab_aes_ccm_init_ext (ATCADevice device, atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_ccm_init (atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_ccm_init_rand_ext (ATCADevice device, atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  Initialize context for AES CCM operation with a random nonce.
- ATCA_STATUS atcab_aes_ccm_init_rand (atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

  Initialize context for AES CCM operation with a random nonce.
- ATCA_STATUS atcab_aes_ccm_aad_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗aad, size_t aad_size)

  Process Additional Authenticated Data (AAD) using CCM mode and a key within the ATECC608A device.
- ATCA_STATUS atcab_aes_ccm_aad_finish (atca_aes_ccm_ctx_t ∗ctx)

  Finish processing Additional Authenticated Data (AAD) using CCM mode.
- ATCA_STATUS atcab_aes_ccm_encrypt_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

  Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ccm_decrypt_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ccm_encrypt_finish (atca_aes_ccm_ctx_t ∗ctx, uint8_t ∗tag, uint8_t ∗tag_size)

  Complete a CCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_ccm_decrypt_finish (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗tag, bool ∗is_verified)

  Complete a CCM decrypt operation authenticating provided tag.
- ATCA_STATUS atcab_pbkdf2_sha256_ext (ATCADevice device, const uint32_t iter, const uint16_t slot, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

  Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.
- ATCA_STATUS atcab_pbkdf2_sha256 (const uint32_t iter, const uint16_t slot, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

  Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.
- ATCA_STATUS _atcab_exit (void)
- ATCA_STATUS atcab_wakeup (void)

  wakeup the CryptoAuth device
- ATCA_STATUS atcab_idle (void)

  idle the CryptoAuth device

- ATCA_STATUS atcab_sleep (void)

    invoke sleep on the CryptoAuth device
- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t *size)

    Gets the size of the specified zone in bytes.
- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t *aes_in, uint8_t *aes_out)

    Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_↩ t *ciphertext)

    Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_encrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_t *ciphertext)

    Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

    Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt_ext (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

    Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_gfm (const uint8_t *h, const uint8_t *input, uint8_t *output)

    Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, const uint8_t *iv, size_t iv_size)

    Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t *free_field, size_t free_field_size, uint8_t *iv)

    Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *aad, uint32_t aad↩ _size)

    Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

    Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t *ctx, uint8_t *tag, size_t tag_size)

    Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *ciphertext, uint32_t ciphertext_size, uint8_t *plaintext)

    Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t *ctx, const uint8_t *tag, size_t tag↩ _size, bool *is_verified)

    Complete a GCM decrypt operation verifying the authentication tag.
- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t *challenge, const uint8_↩ t *response, const uint8_t *other_data)

    Compares a MAC response with input values.
- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t *counter_value)

    Compute the Counter functions.
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t *counter_value)

    Increments one of the device's monotonic counters.
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t *counter_value)

    Read one of the device's monotonic counters.
- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t key_id, const uint8_t *mac)

Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.

- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)

  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id, const uint8_t num_in[(20)])

  ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t ∗other_data, uint8_t other_data↩ _size)

  Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t ∗other_data, uint8_↩ t ∗public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t ∗public_key)

  Issues GenKey command, which generates a new random private key in slot/handle and returns the public key.
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_get_pubkey_ext (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t ∗digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS atcab_info (uint8_t ∗revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS atcab_info_set_latch (bool state)

  Use the Info command to set the persistent latch state for an ATECC608 device.
- ATCA_STATUS atcab_info_get_latch (bool ∗state)

  Use the Info command to get the persistent latch current state for an ATECC608 device.
- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

  Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

  The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS atcab_lock_config_zone (void)

  Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

Lock the config zone with summary CRC.
- ATCA_STATUS atcab_lock_data_zone (void)

  Unconditionally (no CRC required) lock the data zone (slots and OTP). for CryptoAuth devices and lock the setup for Trust Anchor device.
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

  Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

  Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1) (for cryptoauth devices) or Lock an individual handle in shared data element on an Trust Anchor device (for Trust Anchor devices).
- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)

  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[(20)])

  Executes PrivWrite command, to write externally generated ECC private keys into the device.
- ATCA_STATUS atcab_random (uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the device.
- ATCA_STATUS atcab_random_ext (ATCADevice device, uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the device.
- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS atcab_is_config_locked (bool ∗is_locked)

  This function check whether configuration zone is locked or not.
- ATCA_STATUS atcab_is_data_locked (bool ∗is_locked)

  This function check whether data/setup zone is locked or not.
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

  This function check whether slot/handle is locked or not.
- ATCA_STATUS atcab_is_private_ext (ATCADevice device, uint16_t slot, bool ∗is_private)

  Check to see if the key is a private key or not.
- ATCA_STATUS atcab_is_private (uint16_t slot, bool ∗is_private)
- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_↩ t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.

- ATCA_STATUS atcab_read_serial_number (uint8_t *serial_number)

  This function returns serial number of the device.
- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_pubkey_ext (ATCADevice device, uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t *sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS atcab_read_config_zone (uint8_t *config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS atcab_cmp_config_zone (uint8_t *config_data, bool *same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t *result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608 chip.
- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t *data_in, uint8_t *data_out, uint16_t *data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS atcab_sha_start (void)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS atcab_sha_update (const uint8_t *message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS atcab_sha_end (uint8_t *digest, uint16_t length, const uint8_t *message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_read_context (uint8_t *context, uint16_t *context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS atcab_sha_write_context (const uint8_t *context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t *message, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t *data, size_t data_size, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t *ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t *ctx, uint8_t *digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.

- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t ∗ctx, uint16_t key_slot)

  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data↩ _size)

  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t ∗ctx, uint8_t ∗digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac (const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_ext (ATCADevice device, const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t ∗signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t ∗msg, uint8_t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS atcab_sign_ext (ATCADevice device, uint16_t key_id, const uint8_t ∗msg, uint8_↩ t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t ∗signature)

  Executes Sign command to sign an internally generated message.
- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).
- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public↩ _key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS atcab_verify_extern (const uint8_t ∗message, const uint8_t ∗signature, const uint8_↩ t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS atcab_verify_extern_ext (ATCADevice device, const uint8_t ∗message, const uint8_↩ t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS atcab_verify_extern_mac (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.
- ATCA_STATUS atcab_verify_stored (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩ Key for other devices.
- ATCA_STATUS atcab_verify_stored_ext (ATCADevice device, const uint8_t ∗message, const uint8_↩ t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩ Key for other devices.

- ATCA_STATUS atcab_verify_stored_mac (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key←_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_←t ∗data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t ∗public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS atcab_write_config_zone (const uint8_t ∗config_data)

  Executes the Write command, which writes the configuration zone.
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])

  Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

  Initialize one of the monotonic counters in device with a specific value.

Variables

- ATCADevice _gDevice

10.6.1 Detailed Description

CryptoAuthLib Basic API methods - a simple crypto authentication API. These methods manage a global ATCA←Device object behind the scenes. They also manage the wake/idle state transitions so callers don't need to.

Copyright

10.7 atca_bool.h File Reference

bool define for systems that don't have it

```
#include <stdbool.h>
```

10.7.1 Detailed Description

bool define for systems that don't have it

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.8 atca_cfgs.c File Reference

a set of default configurations for various ATCA devices and interfaces

```
#include <stddef.h>
#include "cryptoauthlib.h"
#include "atca_cfgs.h"
#include "atca_iface.h"
#include "atca_device.h"
```

10.8.1 Detailed Description

a set of default configurations for various ATCA devices and interfaces

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.9 atca_cfgs.h File Reference

a set of default configurations for various ATCA devices and interfaces

```
#include "atca_iface.h"
```

Variables

- ATCAIfaceCfg cfg_ateccx08a_i2c_default

  default configuration for an ECCx08A device on the first logical I2C bus
- ATCAIfaceCfg cfg_ateccx08a_swi_default

  default configuration for an ECCx08A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_ateccx08a_kitcdc_default

  default configuration for Kit protocol over a CDC interface
- ATCAIfaceCfg cfg_ateccx08a_kithid_default

  default configuration for Kit protocol over a HID interface
- ATCAIfaceCfg cfg_atsha20xa_i2c_default

  default configuration for a SHA204A device on the first logical I2C bus
- ATCAIfaceCfg cfg_atsha20xa_swi_default

default configuration for an SHA20xA device on the logical SWI bus over UART

- ATCAIfaceCfg cfg_atsha20xa_kitcdc_default

  default configuration for Kit protocol over a CDC interface
- ATCAIfaceCfg cfg_atsha20xa_kithid_default

  default configuration for Kit protocol over a HID interface for SHA204
- ATCAIfaceCfg cfg_ecc204_i2c_default

  default configuration for an ECC204 device on the first logical I2C bus
- ATCAIfaceCfg cfg_ecc204_swi_default

  default configuration for an ECC204 device on the logical SWI over GPIO
- ATCAIfaceCfg cfg_ecc204_kithid_default

  default configuration for Kit protocol over the device's async interface

10.9.1 Detailed Description

a set of default configurations for various ATCA devices and interfaces

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.9.2 Variable Documentation

10.9.2.1 cfg_ateccx08a_i2c_default

```
ATCAIfaceCfg cfg_ateccx08a_i2c_default [extern]
```

default configuration for an ECCx08A device on the first logical I2C bus

10.9.2.2 cfg_ateccx08a_kitcdc_default

```
ATCAIfaceCfg cfg_ateccx08a_kitcdc_default [extern]
```

default configuration for Kit protocol over a CDC interface

10.9.2.3 cfg_ateccx08a_kithid_default

```
ATCAIfaceCfg cfg_ateccx08a_kithid_default [extern]
```

default configuration for Kit protocol over a HID interface

10.9.2.4 cfg_ateccx08a_swi_default

```
ATCAIfaceCfg cfg_ateccx08a_swi_default [extern]
```

default configuration for an ECCx08A device on the logical SWI bus over UART

10.9.2.5 cfg_atsha20xa_i2c_default

```
ATCAIfaceCfg cfg_atsha20xa_i2c_default [extern]
```

default configuration for a SHA204A device on the first logical I2C bus

10.9.2.6 cfg_atsha20xa_kitcdc_default

```
ATCAIfaceCfg cfg_atsha20xa_kitcdc_default [extern]
```

default configuration for Kit protocol over a CDC interface

10.9.2.7 cfg_atsha20xa_kithid_default

```
ATCAIfaceCfg cfg_atsha20xa_kithid_default [extern]
```

default configuration for Kit protocol over a HID interface for SHA204

10.9.2.8 cfg_atsha20xa_swi_default

```
ATCAIfaceCfg cfg_atsha20xa_swi_default [extern]
```

default configuration for an SHA20xA device on the logical SWI bus over UART

10.9.2.9 cfg_ecc204_i2c_default

```
ATCAIfaceCfg cfg_ecc204_i2c_default [extern]
```

default configuration for an ECC204 device on the first logical I2C bus

10.9.2.10 cfg_ecc204_kithid_default

```
ATCAIfaceCfg cfg_ecc204_kithid_default [extern]
```

default configuration for Kit protocol over the device's async interface

10.9.2.11 cfg_ecc204_swi_default

```
ATCAIfaceCfg cfg_ecc204_swi_default [extern]
```

default configuration for an ECC204 device on the logical SWI over GPIO

10.10 atca_compiler.h File Reference

CryptoAuthLiub is meant to be portable across architectures, even non-Microchip architectures and compiler environments. This file is for isolating compiler specific macros.

Macros

- #define SHARED_LIB_EXPORT
- #define ATCA_DLL extern

10.10.1 Detailed Description

CryptoAuthLiub is meant to be portable across architectures, even non-Microchip architectures and compiler environments. This file is for isolating compiler specific macros.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.10.2 Macro Definition Documentation

10.10.2.1 ATCA_DLL

```
#define ATCA_DLL extern
```

10.10.2.2 SHARED_LIB_EXPORT

```
#define SHARED_LIB_EXPORT
```

10.11 atca_crypto_hw_aes.h File Reference

AES CTR, CBC & CMAC structure definitions.

```
#include "cryptoauthlib.h"
```

Data Structures

- struct atca_aes_cbc_ctx
- struct atca_aes_cmac_ctx
- struct atca_aes_ctr_ctx
- struct atca_aes_cbcmac_ctx
- struct atca_aes_ccm_ctx

Typedefs

- typedef struct atca_aes_cbc_ctx atca_aes_cbc_ctx_t
- typedef struct atca_aes_cmac_ctx atca_aes_cmac_ctx_t
- typedef struct atca_aes_ctr_ctx atca_aes_ctr_ctx_t
- typedef struct atca_aes_cbcmac_ctx atca_aes_cbcmac_ctx_t
- typedef struct atca_aes_ccm_ctx atca_aes_ccm_ctx_t

10.11.1 Detailed Description

AES CTR, CBC & CMAC structure definitions.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.11.2 Typedef Documentation

10.11.2.1 atca_aes_cbc_ctx_t

```
typedef struct atca_aes_cbc_ctx atca_aes_cbc_ctx_t
```

10.11.2.2 atca_aes_cbcmac_ctx_t

```
typedef struct atca_aes_cbcmac_ctx atca_aes_cbcmac_ctx_t
```

10.11.2.3 atca_aes_ccm_ctx_t

```
typedef struct atca_aes_ccm_ctx atca_aes_ccm_ctx_t
```

10.11.2.4 atca_aes_cmac_ctx_t

```
typedef struct atca_aes_cmac_ctx atca_aes_cmac_ctx_t
```

10.11.2.5 atca_aes_ctr_ctx_t

```
typedef struct atca_aes_ctr_ctx atca_aes_ctr_ctx_t
```

10.12 atca_crypto_hw_aes_cbc.c File Reference

CryptoAuthLib Basic API methods for AES CBC mode.

```
#include "cryptoauthlib.h"
#include "atca_crypto_hw_aes.h"
```

Functions

- ATCA_STATUS atcab_aes_cbc_init_ext (ATCADevice device, atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

    Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

    Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

    Encrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

    Decrypt a block of data using CBC mode and a key within the device. atcab_aes_cbc_init() should be called before the first use of this function.

10.12.1 Detailed Description

CryptoAuthLib Basic API methods for AES CBC mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode.

Note

List of devices that support this command - ATECC608A, ATECC608B, & TA100. Refer to device datasheet for full details.

Copyright

10.13 atca_crypto_hw_aes_cbcmac.c File Reference

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS atcab_aes_cbcmac_init_ext (ATCADevice device, atca_aes_cbcmac_ctx_t ∗ctx, uint16_↩ t key_id, uint8_t key_block)

  Initialize context for AES CBC-MAC operation.
- ATCA_STATUS atcab_aes_cbcmac_init (atca_aes_cbcmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize context for AES CBC-MAC operation.
- ATCA_STATUS atcab_aes_cbcmac_update (atca_aes_cbcmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_↩ t data_size)

  Calculate AES CBC-MAC with key stored within ECC608A device. calib_aes_cbcmac_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbcmac_finish (atca_aes_cbcmac_ctx_t ∗ctx, uint8_t ∗mac, uint32_t mac_size)

  Finish a CBC-MAC operation returning the CBC-MAC value. If the data provided to the calib_aes_cbcmac_update() function has incomplete block this function will return an error code.

10.13.1 Detailed Description

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

10.14 atca_crypto_hw_aes_ccm.c File Reference

CryptoAuthLib Basic API methods for AES CCM mode.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS atcab_aes_ccm_init_ext (ATCADevice device, atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

    Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_ccm_init (atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

    Initialize context for AES CCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_ccm_init_rand_ext (ATCADevice device, atca_aes_ccm_ctx_t ∗ctx, uint16_↩t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

    Initialize context for AES CCM operation with a random nonce.
- ATCA_STATUS atcab_aes_ccm_init_rand (atca_aes_ccm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t ∗iv, size_t iv_size, size_t aad_size, size_t text_size, size_t tag_size)

    Initialize context for AES CCM operation with a random nonce.
- ATCA_STATUS atcab_aes_ccm_aad_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗aad, size_t aad_↩size)

    Process Additional Authenticated Data (AAD) using CCM mode and a key within the ATECC608A device.
- ATCA_STATUS atcab_aes_ccm_aad_finish (atca_aes_ccm_ctx_t ∗ctx)

    Finish processing Additional Authenticated Data (AAD) using CCM mode.
- ATCA_STATUS atcab_aes_ccm_encrypt_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

    Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_↩init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ccm_decrypt_update (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

    Process data using CCM mode and a key within the ATECC608A device. calib_aes_ccm_init() or calib_aes_ccm_↩init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ccm_encrypt_finish (atca_aes_ccm_ctx_t ∗ctx, uint8_t ∗tag, uint8_t ∗tag_size)

    Complete a CCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_ccm_decrypt_finish (atca_aes_ccm_ctx_t ∗ctx, const uint8_t ∗tag, bool ∗is_↩verified)

    Complete a CCM decrypt operation authenticating provided tag.

10.14.1 Detailed Description

CryptoAuthLib Basic API methods for AES CCM mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. CCM mode provides security and authenticity to the message being processed.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

10.15 atca_crypto_hw_aes_cmac.c File Reference

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

```
#include "cryptoauthlib.h"
#include "atca_crypto_hw_aes.h"
```

Functions

- ATCA_STATUS atcab_aes_cmac_init_ext (ATCADevice device, atca_aes_cmac_ctx_t *ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the device.
- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t *ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the device.
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t *ctx, const uint8_t *data, uint32_t data_↩ size)

  Add data to an initialized CMAC calculation.
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t *ctx, uint8_t *cmac, uint32_t cmac_size)

  Finish a CMAC operation returning the CMAC value.

10.15.1 Detailed Description

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode.

Note

List of devices that support this command - ATECC608A, ATECC608B, & TA100. Refer to device datasheet for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.16 atca_crypto_hw_aes_ctr.c File Reference

CryptoAuthLib Basic API methods for AES CTR mode.

```
#include "cryptoauthlib.h"
#include "atca_crypto_hw_aes.h"
```

Functions

- ATCA_STATUS atcab_aes_ctr_init_ext (ATCADevice device, atca_aes_ctr_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t *iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_↩ t counter_size, const uint8_t *iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand_ext (ATCADevice device, atca_aes_ctr_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t *iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t *ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t *iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t *ctx)

  Increments AES CTR counter value.
- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *input, uint8_t *output)

  Process a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *plaintext, uint8_↩ t *ciphertext)

  Encrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t *ctx, const uint8_t *ciphertext, uint8_↩ t *plaintext)

  Decrypt a block of data using CTR mode and a key within the device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

10.16.1 Detailed Description

CryptoAuthLib Basic API methods for AES CTR mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode.

Note

List of devices that support this command - ATECC608A, ATECC608B, & TA100. Refer to device datasheet for full details.

Copyright

10.17 atca_crypto_pbkdf2.c File Reference

Implementation of the PBKDF2 algorithm for use in generating password hashes.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS atcac_pbkdf2_sha256 (const uint32_t iter, const uint8_t ∗password, const size_t password↵ _len, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

  Calculate a PBKDF2 hash of a given password and salt.
- ATCA_STATUS atcab_pbkdf2_sha256_ext (ATCADevice device, const uint32_t iter, const uint16_t slot, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

  Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.
- ATCA_STATUS atcab_pbkdf2_sha256 (const uint32_t iter, const uint16_t slot, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

  Calculate a PBKDF2 password hash using a stored key inside a device. The key length is determined by the device being used. ECCx08: 32 bytes, TA100: 16-64 bytes.

10.17.1 Detailed Description

Implementation of the PBKDF2 algorithm for use in generating password hashes.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.17.2 Function Documentation

10.17.2.1 atcac_pbkdf2_sha256()

```
ATCA_STATUS atcac_pbkdf2_sha256 (
            const uint32_t iter,
            const uint8_t * password,
            const size_t password_len,
            const uint8_t * salt,
            const size_t salt_len,
            uint8_t * result,
            size_t result_len )
```

Calculate a PBKDF2 hash of a given password and salt.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iter*	Number of iterations of the algorithm to perform
in	*password*	Password to hash
in	*password_len*	Length of the password bytes buffer
in	*salt*	Salt bytes to use
in	*salt_len*	Length of the salt bytes buffer
out	*result*	Output buffer to hold the derived key
in	*result_len*	Length of the key to derive

10.18 atca_crypto_sw.h File Reference

Common defines for CryptoAuthLib software crypto wrappers.

```
#include <stdint.h>
#include <stdlib.h>
#include "atca_config.h"
#include "atca_status.h"
#include "mbedtls/config.h"
#include <mbedtls/cipher.h>
#include <mbedtls/md.h>
#include <mbedtls/pk.h>
```

Macros

- #define ATCA_SHA1_DIGEST_SIZE (20)
- #define ATCA_SHA2_256_DIGEST_SIZE (32)
- #define ATCA_SHA2_256_BLOCK_SIZE (64)
- #define MBEDTLS_CMAC_C

Typedefs

- typedef mbedtls_cipher_context_t atcac_aes_cmac_ctx
- typedef mbedtls_md_context_t atcac_hmac_sha256_ctx
- typedef mbedtls_cipher_context_t atcac_aes_gcm_ctx
- typedef mbedtls_md_context_t atcac_sha1_ctx
- typedef mbedtls_md_context_t atcac_sha2_256_ctx
- typedef mbedtls_pk_context atcac_pk_ctx

Functions

- ATCA_STATUS atcac_aes_gcm_encrypt_start (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len, const uint8_t ∗iv, const uint8_t iv_len)

    Initialize an AES-GCM context.
- ATCA_STATUS atcac_aes_gcm_decrypt_start (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len, const uint8_t ∗iv, const uint8_t iv_len)

    Initialize an AES-GCM context for decryption.
- ATCA_STATUS atcac_aes_cmac_init (atcac_aes_cmac_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len)

    Initialize context for performing CMAC in software.
- ATCA_STATUS atcac_aes_cmac_update (atcac_aes_cmac_ctx ∗ctx, const uint8_t ∗data, const size_↩ t data_size)

    Update CMAC context with input data.
- ATCA_STATUS atcac_aes_cmac_finish (atcac_aes_cmac_ctx ∗ctx, uint8_t ∗cmac, size_t ∗cmac_size)

    Finish CMAC calculation and clear the CMAC context.
- ATCA_STATUS atcac_pk_init (atcac_pk_ctx ∗ctx, uint8_t ∗buf, size_t buflen, uint8_t key_type, bool pubkey)

    Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_init_pem (atcac_pk_ctx ∗ctx, uint8_t ∗buf, size_t buflen, bool pubkey)

    Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_free (atcac_pk_ctx ∗ctx)

    Free a public/private key structure.

- ATCA_STATUS atcac_pk_public (atcac_pk_ctx ∗ctx, uint8_t ∗buf, size_t ∗buflen)

    Get the public key from the context.
- ATCA_STATUS atcac_pk_sign (atcac_pk_ctx ∗ctx, uint8_t ∗digest, size_t dig_len, uint8_t ∗signature, size_t ∗sig_len)

    Perform a signature with the private key in the context.
- ATCA_STATUS atcac_pk_verify (atcac_pk_ctx ∗ctx, uint8_t ∗digest, size_t dig_len, uint8_t ∗signature, size_t sig_len)

    Perform a verify using the public key in the provided context.
- ATCA_STATUS atcac_pk_derive (atcac_pk_ctx ∗private_ctx, atcac_pk_ctx ∗public_ctx, uint8_t ∗buf, size_t ∗buflen)

    Execute the key agreement protocol for the provided keys (if they can)
- ATCA_STATUS atcac_aes_gcm_aad_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗aad, const size_t aad_len)

    Update the GCM context with additional authentication data (AAD)
- ATCA_STATUS atcac_aes_gcm_encrypt_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗plaintext, const size_t pt_len, uint8_t ∗ciphertext, size_t ∗ct_len)

    Encrypt a data using the initialized context.
- ATCA_STATUS atcac_aes_gcm_encrypt_finish (atcac_aes_gcm_ctx ∗ctx, uint8_t ∗tag, size_t tag_len)

    Get the AES-GCM tag and free the context.
- ATCA_STATUS atcac_aes_gcm_decrypt_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗ciphertext, const size_t ct_len, uint8_t ∗plaintext, size_t ∗pt_len)

    Decrypt ciphertext using the initialized context.
- ATCA_STATUS atcac_aes_gcm_decrypt_finish (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗tag, size_t tag_len, bool ∗is_verified)

    Compare the AES-GCM tag and free the context.
- ATCA_STATUS atcac_pbkdf2_sha256 (const uint32_t iter, const uint8_t ∗password, const size_t password_len, const uint8_t ∗salt, const size_t salt_len, uint8_t ∗result, size_t result_len)

    Calculate a PBKDF2 hash of a given password and salt.

10.18.1 Detailed Description

Common defines for CryptoAuthLib software crypto wrappers.

Copyright

10.18.2 Macro Definition Documentation

10.18.2.1 ATCA_SHA1_DIGEST_SIZE

```
#define ATCA_SHA1_DIGEST_SIZE (20)
```

10.18.2.2 ATCA_SHA2_256_BLOCK_SIZE

```
#define ATCA_SHA2_256_BLOCK_SIZE (64)
```

10.18.2.3 ATCA_SHA2_256_DIGEST_SIZE

```
#define ATCA_SHA2_256_DIGEST_SIZE (32)
```

10.18.2.4 MBEDTLS_CMAC_C

```
#define MBEDTLS_CMAC_C
```

10.18.3 Typedef Documentation

10.18.3.1 atcac_aes_cmac_ctx

```
typedef mbedtls_cipher_context_t atcac_aes_cmac_ctx
```

10.18.3.2 atcac_aes_gcm_ctx

```
typedef mbedtls_cipher_context_t atcac_aes_gcm_ctx
```

10.18.3.3 atcac_hmac_sha256_ctx

```
typedef mbedtls_md_context_t atcac_hmac_sha256_ctx
```

10.18.3.4 atcac_pk_ctx

```
typedef mbedtls_pk_context atcac_pk_ctx
```

10.18.3.5 atcac_sha1_ctx

```
typedef mbedtls_md_context_t atcac_sha1_ctx
```

10.18.3.6 atcac_sha2_256_ctx

```
typedef mbedtls_md_context_t atcac_sha2_256_ctx
```

10.18.4 Function Documentation

10.18.4.1 atcac_aes_cmac_finish()

```
ATCA_STATUS atcac_aes_cmac_finish (
            atcac_aes_cmac_ctx * ctx,
            uint8_t * cmac,
            size_t * cmac_size )
```

Finish CMAC calculation and clear the CMAC context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
out	*cmac*	cmac value
in,out	*cmac_size*	length of cmac

10.18.4.2 atcac_aes_cmac_init()

```
ATCA_STATUS atcac_aes_cmac_init (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len )
```

Initialize context for performing CMAC in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*key*	key value to use
in	*key_len*	length of the key

10.18.4.3 atcac_aes_cmac_update()

```
ATCA_STATUS atcac_aes_cmac_update (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * data,
            const size_t data_size )
```

Update CMAC context with input data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*data*	input data
in	*data_size*	length of input data

10.18.4.4 atcac_aes_gcm_aad_update()

```
ATCA_STATUS atcac_aes_gcm_aad_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * aad,
            const size_t aad_len )
```

Update the GCM context with additional authentication data (AAD)

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*aad*	Additional Authentication Data
in	*aad_len*	Length of AAD

10.18.4.5 atcac_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_decrypt_finish (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * tag,
            size_t tag_len,
            bool * is_verified )
```

Compare the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*tag*	GCM Tag to Verify
in	*tag_len*	Length of the GCM tag
out	*is_verified*	Tag verified as matching

10.18.4.6 atcac_aes_gcm_decrypt_start()

```
ATCA_STATUS atcac_aes_gcm_decrypt_start (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len,
            const uint8_t * iv,
            const uint8_t iv_len )
```

Initialize an AES-GCM context for decryption.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.18.4.7 atcac_aes_gcm_decrypt_update()

```
ATCA_STATUS atcac_aes_gcm_decrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * ciphertext,
            const size_t ct_len,
            uint8_t * plaintext,
            size_t * pt_len )
```

Decrypt ciphertext using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*ciphertext*	Ciphertext to decrypt
in	*ct_len*	Length of the ciphertext
out	*plaintext*	Resulting decrypted plaintext
in,out	*pt_len*	Length of the plaintext buffer

10.18.4.8 atcac_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_encrypt_finish (
            atcac_aes_gcm_ctx * ctx,
            uint8_t * tag,
            size_t tag_len )
```

Get the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
out	*tag*	GCM Tag Result
in	*tag_len*	Length of the GCM tag

10.18.4.9 atcac_aes_gcm_encrypt_start()

```
ATCA_STATUS atcac_aes_gcm_encrypt_start (
            atcac_aes_gcm_ctx * ctx,
```

```
            const uint8_t * key,
            const uint8_t key_len,
            const uint8_t * iv,
            const uint8_t iv_len )
```

Initialize an AES-GCM context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.18.4.10 atcac_aes_gcm_encrypt_update()

```
ATCA_STATUS atcac_aes_gcm_encrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * plaintext,
            const size_t pt_len,
            uint8_t * ciphertext,
            size_t * ct_len )
```

Encrypt a data using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*plaintext*	Input buffer to encrypt
in	*pt_len*	Length of the input
out	*ciphertext*	Output buffer
in,out	*ct_len*	Length of the ciphertext buffer

10.18.4.11 atcac_pbkdf2_sha256()

```
ATCA_STATUS atcac_pbkdf2_sha256 (
            const uint32_t iter,
```

```
            const uint8_t * password,
            const size_t password_len,
            const uint8_t * salt,
            const size_t salt_len,
            uint8_t * result,
            size_t result_len )
```

Calculate a PBKDF2 hash of a given password and salt.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iter*	Number of iterations of the algorithm to perform
in	*password*	Password to hash
in	*password_len*	Length of the password bytes buffer
in	*salt*	Salt bytes to use
in	*salt_len*	Length of the salt bytes buffer
out	*result*	Output buffer to hold the derived key
in	*result_len*	Length of the key to derive

10.18.4.12 atcac_pk_derive()

```
ATCA_STATUS atcac_pk_derive (
            atcac_pk_ctx * private_ctx,
            atcac_pk_ctx * public_ctx,
            uint8_t * buf,
            size_t * buflen )
```

Execute the key agreement protocol for the provided keys (if they can)

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.18.4.13 atcac_pk_free()

```
ATCA_STATUS atcac_pk_free (
            atcac_pk_ctx * ctx )
```

Free a public/private key structure.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context

10.18.4.14 atcac_pk_init()

```
ATCA_STATUS atcac_pk_init (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t buflen,
            uint8_t key_type,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.18.4.15 atcac_pk_init_pem()

```
ATCA_STATUS atcac_pk_init_pem (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t buflen,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.18.4.16 atcac_pk_public()

```
ATCA_STATUS atcac_pk_public (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t * buflen )
```

Get the public key from the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.18.4.17 atcac_pk_sign()

```
ATCA_STATUS atcac_pk_sign (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t * sig_len )
```

Perform a signature with the private key in the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.18.4.18 atcac_pk_verify()

```
ATCA_STATUS atcac_pk_verify (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t sig_len )
```

Perform a verify using the public key in the provided context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.19 atca_crypto_sw_ecdsa.c File Reference

API wrapper for software ECDSA verify. Currently unimplemented but could be implemented via a 3rd party library such as MicroECC.

```
#include "atca_crypto_sw_ecdsa.h"
```

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[(256/8)], const uint8_t signature[((256/8) *2)], const uint8_t public_key[((256/8) *2)])

  return software generated ECDSA verification result and the function is currently not implemented

10.19.1 Detailed Description

API wrapper for software ECDSA verify. Currently unimplemented but could be implemented via a 3rd party library such as MicroECC.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.20 atca_crypto_sw_ecdsa.h File Reference

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Macros

- #define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
- #define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
- #define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
- #define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[(256/8)], const uint8_t signature[((256/8) *2)], const uint8_t public_key[((256/8) *2)])

  return software generated ECDSA verification result and the function is currently not implemented

10.20.1 Detailed Description

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.21 atca_crypto_sw_rand.c File Reference

API wrapper for software random.

```
#include "cryptoauthlib.h"
```

10.21.1 Detailed Description

API wrapper for software random.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.22 atca_crypto_sw_rand.h File Reference

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Functions

- int atcac_sw_random (uint8_t *data, size_t data_size)

  Return Random Bytes.

10.22.1 Detailed Description

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.23 atca_crypto_sw_sha1.c File Reference

Wrapper API for SHA 1 routines.

```
#include "atca_crypto_sw_sha1.h"
#include "hashes/sha1_routines.h"
```

Functions

- int atcac_sw_sha1 (const uint8_t *data, size_t data_size, uint8_t digest[(20)])

  Perform SHA1 hash of data in software.

10.23.1 Detailed Description

Wrapper API for SHA 1 routines.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.24 atca_crypto_sw_sha1.h File Reference

Wrapper API for SHA 1 routines.

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Functions

- int atcac_sw_sha1_init (atcac_sha1_ctx ∗ctx)

  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Add data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx ∗ctx, uint8_t digest[(20)])
- int atcac_sw_sha1 (const uint8_t ∗data, size_t data_size, uint8_t digest[(20)])

  Perform SHA1 hash of data in software.

10.24.1 Detailed Description

Wrapper API for SHA 1 routines.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.25 atca_crypto_sw_sha2.c File Reference

Wrapper API for software SHA 256 routines.

```
#include "cryptoauthlib.h"
#include "atca_crypto_sw_sha2.h"
#include "hashes/sha2_routines.h"
```

Functions

- int atcac_sw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t digest[(32)])

  single call convenience function which computes Hash of given data using SHA256 software
- ATCA_STATUS atcac_sha256_hmac_counter (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗label, size_t label_↩ len, uint8_t ∗data, size_t data_len, uint8_t ∗digest, size_t diglen)

  Implements SHA256 HMAC-Counter per NIST SP 800-108 used for KDF like operations.

10.25.1 Detailed Description

Wrapper API for software SHA 256 routines.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.26 atca_crypto_sw_sha2.h File Reference

Wrapper API for software SHA 256 routines.

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Functions

- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx ∗ctx)

  Initialize context for performing SHA256 hash in software.
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Add data to a SHA256 hash.
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx ∗ctx, uint8_t digest[(32)])
- int atcac_sw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t digest[(32)])

  single call convenience function which computes Hash of given data using SHA256 software
- ATCA_STATUS atcac_sha256_hmac_init (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len)

  Initialize context for performing HMAC (sha256) in software.
- ATCA_STATUS atcac_sha256_hmac_update (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗data, size_↩ t data_size)

  Update HMAC context with input data.
- ATCA_STATUS atcac_sha256_hmac_finish (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗digest, size_t ∗digest↩ _len)

  Finish CMAC calculation and clear the HMAC context.
- ATCA_STATUS atcac_sha256_hmac_counter (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗label, size_t label_↩ len, uint8_t ∗data, size_t data_len, uint8_t ∗digest, size_t diglen)

  Implements SHA256 HMAC-Counter per NIST SP 800-108 used for KDF like operations.

10.26.1 Detailed Description

Wrapper API for software SHA 256 routines.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.27 atca_debug.c File Reference

Debug/Trace for CryptoAuthLib calls.

```
#include <cryptoauthlib.h>
```

Functions

- void atca_trace_config (FILE *fp)
- ATCA_STATUS atca_trace (ATCA_STATUS status)
- ATCA_STATUS atca_trace_msg (ATCA_STATUS status, const char *msg)

Variables

- FILE * g_trace_fp

10.27.1 Detailed Description

Debug/Trace for CryptoAuthLib calls.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.27.2 Function Documentation

10.27.2.1 atca_trace()

```
ATCA_STATUS atca_trace (
            ATCA_STATUS status )
```

10.27.2.2 atca_trace_config()

```
void atca_trace_config (
            FILE * fp )
```

10.27.2.3 atca_trace_msg()

```
ATCA_STATUS atca_trace_msg (
            ATCA_STATUS status,
            const char * msg )
```

10.27.3 Variable Documentation

10.27.3.1 g_trace_fp

```
FILE* g_trace_fp
```

10.28 atca_debug.h File Reference

```
#include "atca_status.h"
```

Functions

- void atca_trace_config (FILE *fp)
- ATCA_STATUS atca_trace (ATCA_STATUS status)
- ATCA_STATUS atca_trace_msg (ATCA_STATUS status, const char *msg)

10.28.1 Function Documentation

10.28.1.1 atca_trace()

```
ATCA_STATUS atca_trace (
            ATCA_STATUS status )
```

10.28.1.2 atca_trace_config()

```
void atca_trace_config (
            FILE * fp )
```

10.28.1.3 atca_trace_msg()

```
ATCA_STATUS atca_trace_msg (
            ATCA_STATUS status,
            const char * msg )
```

10.29 atca_device.c File Reference

Microchip CryptoAuth device object.

```
#include <cryptoauthlib.h>
```

Functions

- ATCADevice newATCADevice (ATCAIfaceCfg *cfg)

  constructor for a Microchip CryptoAuth device
- void deleteATCADevice (ATCADevice *ca_dev)

  destructor for a device NULLs reference after object is freed
- ATCA_STATUS initATCADevice (ATCAIfaceCfg *cfg, ATCADevice ca_dev)

  Initializer for an Microchip CryptoAuth device.
- ATCAIface atGetIFace (ATCADevice dev)

  returns a reference to the ATCAIface interface object for the device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

  Release any resources associated with the device.

10.29.1 Detailed Description

Microchip CryptoAuth device object.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.30 atca_device.h File Reference

Microchip Crypto Auth device object.

```
#include "atca_iface.h"
```

Data Structures

- struct _atsha204a_config
- struct _atecc508a_config
- struct _atecc608_config
- struct atca_device

    atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods

Macros

- #define ATCA_PACKED
- #define ATCA_AES_ENABLE_EN_SHIFT (0)
- #define ATCA_AES_ENABLE_EN_MASK (0x01u << ATCA_AES_ENABLE_EN_SHIFT)
- #define ATCA_I2C_ENABLE_EN_SHIFT (0)
- #define ATCA_I2C_ENABLE_EN_MASK (0x01u << ATCA_I2C_ENABLE_EN_SHIFT)
- #define ATCA_COUNTER_MATCH_EN_SHIFT (0)
- #define ATCA_COUNTER_MATCH_EN_MASK (0x01u << ATCA_COUNTER_MATCH_EN_SHIFT)
- #define ATCA_COUNTER_MATCH_KEY_SHIFT (4)
- #define ATCA_COUNTER_MATCH_KEY_MASK (0x0Fu << ATCA_COUNTER_MATCH_KEY_SHIFT)
- #define ATCA_COUNTER_MATCH_KEY(v) (ATCA_COUNTER_MATCH_KEY_MASK & (v << ATCA_COUNTER_MATCH_K
- #define ATCA_CHIP_MODE_I2C_EXTRA_SHIFT (0)
- #define ATCA_CHIP_MODE_I2C_EXTRA_MASK (0x01u << ATCA_CHIP_MODE_I2C_EXTRA_SHIFT)
- #define ATCA_CHIP_MODE_TTL_EN_SHIFT (1)
- #define ATCA_CHIP_MODE_TTL_EN_MASK (0x01u << ATCA_CHIP_MODE_TTL_EN_SHIFT)
- #define ATCA_CHIP_MODE_WDG_LONG_SHIFT (2)
- #define ATCA_CHIP_MODE_WDG_LONG_MASK (0x01u << ATCA_CHIP_MODE_WDG_LONG_SHIFT)
- #define ATCA_CHIP_MODE_CLK_DIV_SHIFT (3)
- #define ATCA_CHIP_MODE_CLK_DIV_MASK (0x1Fu << ATCA_CHIP_MODE_CLK_DIV_SHIFT)
- #define ATCA_CHIP_MODE_CLK_DIV(v) (ATCA_CHIP_MODE_CLK_DIV_MASK & (v << ATCA_CHIP_MODE_CLK_DIV_S
- #define ATCA_SLOT_CONFIG_READKEY_SHIFT (0)
- #define ATCA_SLOT_CONFIG_READKEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_READKEY_SHIFT)
- #define ATCA_SLOT_CONFIG_READKEY(v) (ATCA_SLOT_CONFIG_READKEY_MASK & (v << ATCA_SLOT_CONFIG_READKEY_SHIFT))
- #define ATCA_SLOT_CONFIG_NOMAC_SHIFT (4)
- #define ATCA_SLOT_CONFIG_NOMAC_MASK (0x01u << ATCA_SLOT_CONFIG_NOMAC_SHIFT)
- #define ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT (5)
- #define ATCA_SLOT_CONFIG_LIMITED_USE_MASK (0x01u << ATCA_SLOT_CONFIG_LIMITED_USE_SHIFT)
- #define ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIFT (6)
- #define ATCA_SLOT_CONFIG_ENCRYPTED_READ_MASK (0x01u << ATCA_SLOT_CONFIG_ENCRYPTED_READ_SHIF
- #define ATCA_SLOT_CONFIG_IS_SECRET_SHIFT (7)
- #define ATCA_SLOT_CONFIG_IS_SECRET_MASK (0x01u << ATCA_SLOT_CONFIG_IS_SECRET_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT (8)
- #define ATCA_SLOT_CONFIG_WRITE_KEY_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_KEY(v) (ATCA_SLOT_CONFIG_WRITE_KEY_MASK & (v << ATCA_SLOT_CONFIG_WRITE_KEY_SHIFT))
- #define ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT (12)
- #define ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK (0x0Fu << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_CONFIG(v) (ATCA_SLOT_CONFIG_WRITE_CONFIG_MASK & (v << ATCA_SLOT_CONFIG_WRITE_CONFIG_SHIFT))
- #define ATCA_SLOT_CONFIG_EXT_SIG_SHIFT (0)
- #define ATCA_SLOT_CONFIG_EXT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_EXT_SIG_SHIFT)
- #define ATCA_SLOT_CONFIG_INT_SIG_SHIFT (1)
- #define ATCA_SLOT_CONFIG_INT_SIG_MASK (0x01u << ATCA_SLOT_CONFIG_INT_SIG_SHIFT)
- #define ATCA_SLOT_CONFIG_ECDH_SHIFT (2)

- #define ATCA_SLOT_CONFIG_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_ECDH_SHIFT)
- #define ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT (3)
- #define ATCA_SLOT_CONFIG_WRITE_ECDH_MASK (0x01u << ATCA_SLOT_CONFIG_WRITE_ECDH_SHIFT)
- #define ATCA_SLOT_CONFIG_GEN_KEY_SHIFT (8)
- #define ATCA_SLOT_CONFIG_GEN_KEY_MASK (0x01u << ATCA_SLOT_CONFIG_GEN_KEY_SHIFT)
- #define ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT (9)
- #define ATCA_SLOT_CONFIG_PRIV_WRITE_MASK (0x01u << ATCA_SLOT_CONFIG_PRIV_WRITE_SHIFT)
- #define ATCA_USE_LOCK_ENABLE_SHIFT (0)
- #define ATCA_USE_LOCK_ENABLE_MASK (0x0Fu << ATCA_USE_LOCK_ENABLE_SHIFT)
- #define ATCA_USE_LOCK_KEY_SHIFT (4)
- #define ATCA_USE_LOCK_KEY_MASK (0x0Fu << ATCA_USE_LOCK_KEY_SHIFT)
- #define ATCA_VOL_KEY_PERM_SLOT_SHIFT (0)
- #define ATCA_VOL_KEY_PERM_SLOT_MASK (0x0Fu << ATCA_VOL_KEY_PERM_SLOT_SHIFT)
- #define ATCA_VOL_KEY_PERM_SLOT(v) (ATCA_VOL_KEY_PERM_SLOT_MASK & (v << ATCA_VOL_KEY_PERM_SLOT
- #define ATCA_VOL_KEY_PERM_EN_SHIFT (7)
- #define ATCA_VOL_KEY_PERM_EN_MASK (0x01u << ATCA_VOL_KEY_PERM_EN_SHIFT)
- #define ATCA_SECURE_BOOT_MODE_SHIFT (0)
- #define ATCA_SECURE_BOOT_MODE_MASK (0x03u << ATCA_SECURE_BOOT_MODE_SHIFT)
- #define ATCA_SECURE_BOOT_MODE(v) (ATCA_SECURE_BOOT_MODE_MASK & (v << ATCA_SECURE_BOOT_MODE
- #define ATCA_SECURE_BOOT_PERSIST_EN_SHIFT (3)
- #define ATCA_SECURE_BOOT_PERSIST_EN_MASK (0x01u << ATCA_SECURE_BOOT_PERSIST_EN_SHIFT)
- #define ATCA_SECURE_BOOT_RAND_NONCE_SHIFT (4)
- #define ATCA_SECURE_BOOT_RAND_NONCE_MASK (0x01u << ATCA_SECURE_BOOT_RAND_NONCE_SHIFT)
- #define ATCA_SECURE_BOOT_DIGEST_SHIFT (8)
- #define ATCA_SECURE_BOOT_DIGEST_MASK (0x0Fu << ATCA_SECURE_BOOT_DIGEST_SHIFT)
- #define ATCA_SECURE_BOOT_DIGEST(v) (ATCA_SECURE_BOOT_DIGEST_MASK & (v << ATCA_SECURE_BOOT_DIG
- #define ATCA_SECURE_BOOT_PUB_KEY_SHIFT (12)
- #define ATCA_SECURE_BOOT_PUB_KEY_MASK (0x0Fu << ATCA_SECURE_BOOT_PUB_KEY_SHIFT)
- #define ATCA_SECURE_BOOT_PUB_KEY(v) (ATCA_SECURE_BOOT_PUB_KEY_MASK & (v << ATCA_SECURE_BOOT_PUB_KEY_SHIFT))
- #define ATCA_SLOT_LOCKED(v) ((0x01 << v) & 0xFFFFu)
- #define ATCA_CHIP_OPT_POST_EN_SHIFT (0)
- #define ATCA_CHIP_OPT_POST_EN_MASK (0x01u << ATCA_CHIP_OPT_POST_EN_SHIFT)
- #define ATCA_CHIP_OPT_IO_PROT_EN_SHIFT (1)
- #define ATCA_CHIP_OPT_IO_PROT_EN_MASK (0x01u << ATCA_CHIP_OPT_IO_PROT_EN_SHIFT)
- #define ATCA_CHIP_OPT_KDF_AES_EN_SHIFT (2)
- #define ATCA_CHIP_OPT_KDF_AES_EN_MASK (0x01u << ATCA_CHIP_OPT_KDF_AES_EN_SHIFT)
- #define ATCA_CHIP_OPT_ECDH_PROT_SHIFT (8)
- #define ATCA_CHIP_OPT_ECDH_PROT_MASK (0x03u << ATCA_CHIP_OPT_ECDH_PROT_SHIFT)
- #define ATCA_CHIP_OPT_ECDH_PROT(v) (ATCA_CHIP_OPT_ECDH_PROT_MASK & (v << ATCA_CHIP_OPT_ECDH_PF
- #define ATCA_CHIP_OPT_KDF_PROT_SHIFT (10)
- #define ATCA_CHIP_OPT_KDF_PROT_MASK (0x03u << ATCA_CHIP_OPT_KDF_PROT_SHIFT)
- #define ATCA_CHIP_OPT_KDF_PROT(v) (ATCA_CHIP_OPT_KDF_PROT_MASK & (v << ATCA_CHIP_OPT_KDF_PROT_
- #define ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT (12)
- #define ATCA_CHIP_OPT_IO_PROT_KEY_MASK (0x0Fu << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT)
- #define ATCA_CHIP_OPT_IO_PROT_KEY(v) (ATCA_CHIP_OPT_IO_PROT_KEY_MASK & (v << ATCA_CHIP_OPT_IO_PROT_KEY_SHIFT))
- #define ATCA_KEY_CONFIG_OFFSET(x) (96UL + (x) * 2)
- #define ATCA_KEY_CONFIG_PRIVATE_SHIFT (0)
- #define ATCA_KEY_CONFIG_PRIVATE_MASK (0x01u << ATCA_KEY_CONFIG_PRIVATE_SHIFT)
- #define ATCA_KEY_CONFIG_PUB_INFO_SHIFT (1)
- #define ATCA_KEY_CONFIG_PUB_INFO_MASK (0x01u << ATCA_KEY_CONFIG_PUB_INFO_SHIFT)
- #define ATCA_KEY_CONFIG_KEY_TYPE_SHIFT (2)
- #define ATCA_KEY_CONFIG_KEY_TYPE_MASK (0x07u << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT)

- #define ATCA_KEY_CONFIG_KEY_TYPE(v) (ATCA_KEY_CONFIG_KEY_TYPE_MASK & (v << ATCA_KEY_CONFIG_KEY_TYPE_SHIFT))
- #define ATCA_KEY_CONFIG_LOCKABLE_SHIFT (5)
- #define ATCA_KEY_CONFIG_LOCKABLE_MASK (0x01u << ATCA_KEY_CONFIG_LOCKABLE_SHIFT)
- #define ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT (6)
- #define ATCA_KEY_CONFIG_REQ_RANDOM_MASK (0x01u << ATCA_KEY_CONFIG_REQ_RANDOM_SHIFT)
- #define ATCA_KEY_CONFIG_REQ_AUTH_SHIFT (7)
- #define ATCA_KEY_CONFIG_REQ_AUTH_MASK (0x01u << ATCA_KEY_CONFIG_REQ_AUTH_SHIFT)
- #define ATCA_KEY_CONFIG_AUTH_KEY_SHIFT (8)
- #define ATCA_KEY_CONFIG_AUTH_KEY_MASK (0x0Fu << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT)
- #define ATCA_KEY_CONFIG_AUTH_KEY(v) (ATCA_KEY_CONFIG_AUTH_KEY_MASK & (v << ATCA_KEY_CONFIG_AUTH_KEY_SHIFT))
- #define ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT (12)
- #define ATCA_KEY_CONFIG_PERSIST_DISABLE_MASK (0x01u << ATCA_KEY_CONFIG_PERSIST_DISABLE_SHIFT)
- #define ATCA_KEY_CONFIG_RFU_SHIFT (13)
- #define ATCA_KEY_CONFIG_RFU_MASK (0x01u << ATCA_KEY_CONFIG_RFU_SHIFT)
- #define ATCA_KEY_CONFIG_X509_ID_SHIFT (14)
- #define ATCA_KEY_CONFIG_X509_ID_MASK (0x03u << ATCA_KEY_CONFIG_X509_ID_SHIFT)
- #define ATCA_KEY_CONFIG_X509_ID(v) (ATCA_KEY_CONFIG_X509_ID_MASK & (v << ATCA_KEY_CONFIG_X509_ID_S

Typedefs

- typedef struct _atsha204a_config atsha204a_config_t
- typedef struct _atecc508a_config atecc508a_config_t
- typedef struct _atecc608_config atecc608_config_t
- typedef struct atca_device ∗ ATCADevice

Enumerations

- enum ATCADeviceState { ATCA_DEVICE_STATE_UNKNOWN = 0, ATCA_DEVICE_STATE_SLEEP, ATCA_DEVICE_STATE_IDLE, ATCA_DEVICE_STATE_ACTIVE }

  ATCADeviceState says about device state.

Functions

- ATCA_STATUS initATCADevice (ATCAIfaceCfg ∗cfg, ATCADevice ca_dev)

  Initializer for an Microchip CryptoAuth device.
- ATCADevice newATCADevice (ATCAIfaceCfg ∗cfg)

  constructor for a Microchip CryptoAuth device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

  Release any resources associated with the device.
- void deleteATCADevice (ATCADevice ∗ca_dev)

  destructor for a device NULLs reference after object is freed
- ATCAIface atGetIFace (ATCADevice dev)

  returns a reference to the ATCAIface interface object for the device

10.30.1 Detailed Description

Microchip Crypto Auth device object.

Copyright

10.31 atca_devtypes.h File Reference

Microchip Crypto Auth.

Enumerations

- enum ATCADeviceType {
  ATSHA204A = 0, ATECC108A = 1, ATECC508A = 2, ATECC608A = 3,
  ATECC608B = 3, ATECC608 = 3, ATSHA206A = 4, ECC204 = 5,
  TA100 = 0x10, ATCA_DEV_UNKNOWN = 0x20 }

  The supported Device type in Cryptoauthlib library.

10.31.1 Detailed Description

Microchip Crypto Auth.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.32 atca_hal.c File Reference

low-level HAL - methods used to setup indirection to physical layer interface. this level does the dirty work of abstracting the higher level ATCAIFace methods from the low-level physical interfaces. Its main goal is to keep low-level details from bleeding into the logical interface implemetation.

```
#include "cryptoauthlib.h"
#include "atca_hal.h"
```

Data Structures

- struct atca_hal_list_entry_t

  Structure that holds the hal/phy maping for different interface types.

Macros

- #define ATCA_MAX_HAL_CACHE

Functions

- ATCA_STATUS hal_iface_register_hal (ATCAIfaceType iface_type, ATCAHAL_t ∗hal, ATCAHAL_t ∗∗old←_hal, ATCAHAL_t ∗phy, ATCAHAL_t ∗∗old_phy)

  Register/Replace a HAL with a.
- ATCA_STATUS hal_iface_init (ATCAIfaceCfg ∗cfg, ATCAHAL_t ∗∗hal, ATCAHAL_t ∗∗phy)

  Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType iface_type, void ∗hal_data)

  releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t ∗response, int response_size)

  Utility function for hal_wake to check the reply.
- uint8_t hal_is_command_word (uint8_t word_address)

  Utility function for hal_wake to check the reply.

10.32.1 Detailed Description

low-level HAL - methods used to setup indirection to physical layer interface. this level does the dirty work of abstracting the higher level ATCAIFace methods from the low-level physical interfaces. Its main goal is to keep low-level details from bleeding into the logical interface implemetation.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.32.2 Macro Definition Documentation

10.32.2.1 ATCA_MAX_HAL_CACHE

```
#define ATCA_MAX_HAL_CACHE
```

10.33 atca_hal.h File Reference

low-level HAL - methods used to setup indirection to physical layer interface

```
#include <stdlib.h>
#include "atca_config.h"
#include "atca_status.h"
#include "atca_iface.h"
```

Data Structures

- struct atca_hal_kit_phy_t

Macros

- #define ATCA_POLLING_INIT_TIME_MSEC 1
- #define ATCA_POLLING_FREQUENCY_TIME_MSEC 2
- #define ATCA_POLLING_MAX_TIME_MSEC 2500
- #define hal_memset_s atcab_memset_s

Enumerations

- enum ATCA_HAL_CONTROL {
ATCA_HAL_CONTROL_WAKE = 0, ATCA_HAL_CONTROL_IDLE = 1, ATCA_HAL_CONTROL_SLEEP = 2, ATCA_HAL_CONTROL_RESET = 3,
ATCA_HAL_CONTROL_SELECT = 4, ATCA_HAL_CONTROL_DESELECT = 5, ATCA_HAL_CHANGE_BAUD = 6, ATCA_HAL_FLUSH_BUFFER = 7 }

Functions

- ATCA_STATUS hal_iface_init (ATCAIfaceCfg *, ATCAHAL_t **hal, ATCAHAL_t **phy)

  Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType, void *hal_data)

  releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t *response, int response_size)

  Utility function for hal_wake to check the reply.
- void atca_delay_ms (uint32_t ms)

  Timer API for legacy implementations.
- void atca_delay_us (uint32_t delay)

  This function delays for a number of microseconds.
- void hal_rtos_delay_ms (uint32_t ms)

  Timer API implemented at the HAL level.
- void hal_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.
- void hal_delay_us (uint32_t delay)

  This function delays for a number of microseconds.
- ATCA_STATUS hal_create_mutex (void **ppMutex, char *pName)

  Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void *pMutex)
- ATCA_STATUS hal_lock_mutex (void *pMutex)
- ATCA_STATUS hal_unlock_mutex (void *pMutex)
- void * hal_malloc (size_t size)
- void hal_free (void *ptr)
- ATCA_STATUS hal_iface_register_hal (ATCAIfaceType iface_type, ATCAHAL_t *hal, ATCAHAL_t **old_hal, ATCAHAL_t *phy, ATCAHAL_t **old_phy)

  Register/Replace a HAL with a.
- uint8_t hal_is_command_word (uint8_t word_address)

  Utility function for hal_wake to check the reply.

10.33.1 Detailed Description

low-level HAL - methods used to setup indirection to physical layer interface

Copyright

10.34 atca_helpers.c File Reference

Helpers to support the CryptoAuthLib Basic API methods.

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "cryptoauthlib.h"
#include "atca_helpers.h"
```

Macros

- #define B64_IS_EQUAL (uint8_t)64
- #define B64_IS_INVALID (uint8_t)0xFF

Functions

- ATCA_STATUS atcab_bin2hex (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size)

  Convert a binary buffer to a hex string for easy reading.
- ATCA_STATUS atcab_reversal (const uint8_t ∗bin, size_t bin_size, uint8_t ∗dest, size_t ∗dest_size)

  To reverse the input data.
- ATCA_STATUS atcab_bin2hex_ (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size, bool is_pretty, bool is_space, bool is_upper)

  Function that converts a binary buffer to a hex string suitable for easy reading.
- ATCA_STATUS atcab_hex2bin_ (const char ∗hex, size_t hex_size, uint8_t ∗bin, size_t ∗bin_size, bool is_space)
- ATCA_STATUS atcab_hex2bin (const char ∗hex, size_t hex_size, uint8_t ∗bin, size_t ∗bin_size)

  Function that converts a hex string to binary buffer.
- bool isDigit (char c)

  Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))
- bool isBlankSpace (char c)

  Checks to see if a character is blank space.
- bool isAlpha (char c)

  Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHexAlpha (char c)

  Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHex (char c)

  Returns true if this character is a valid hex character or if this is blankspace (The character can be included in a valid hexstring).
- bool isHexDigit (char c)

  Returns true if this character is a valid hex character.
- ATCA_STATUS packHex (const char ∗ascii_hex, size_t ascii_hex_len, char ∗packed_hex, size_t ∗packed_len)

  Remove spaces from a ASCII hex string.
- bool isBase64 (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character or if this is space (A character can be included in a valid base 64 string).
- bool isBase64Digit (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character.
- uint8_t base64Index (char c, const uint8_t ∗rules)

  Returns the base 64 index of the given character.
- char base64Char (uint8_t id, const uint8_t ∗rules)

  Returns the base 64 character of the given index.
- ATCA_STATUS atcab_base64decode_ (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_t ∗data_size, const uint8_t ∗rules)

  Decode base64 string to data with ruleset option.
- ATCA_STATUS atcab_base64encode_ (const uint8_t ∗data, size_t data_size, char ∗encoded, size_t ∗encoded_size, const uint8_t ∗rules)

  Encode data as base64 string with ruleset option.
- ATCA_STATUS atcab_base64encode (const uint8_t ∗byte_array, size_t array_len, char ∗encoded, size_t ∗encoded_len)

  Encode data as base64 string.

- ATCA_STATUS atcab_base64decode (const char *encoded, size_t encoded_len, uint8_t *byte_array, size↵ _t *array_len)

    Decode base64 string to data.
- int atcab_memset_s (void *dest, size_t destsz, int ch, size_t count)

    Guaranteed to perform memory writes regardless of optimization level. Matches memset_s signature.

Variables

- uint8_t atcab_b64rules_default [4] = { '+', '/', '=', 64 }
- uint8_t atcab_b64rules_mime [4] = { '+', '/', '=', 76 }
- uint8_t atcab_b64rules_urlsafe [4] = { '-', '_', 0, 0 }

10.34.1 Detailed Description

Helpers to support the CryptoAuthLib Basic API methods.

Copyright

10.34.2 Macro Definition Documentation

10.34.2.1 B64_IS_EQUAL

```
#define B64_IS_EQUAL (uint8_t)64
```

10.34.2.2 B64_IS_INVALID

```
#define B64_IS_INVALID (uint8_t)0xFF
```

10.34.3 Function Documentation

10.34.3.1 atcab_base64decode()

```
ATCA_STATUS atcab_base64decode (
            const char * encoded,
            size_t encoded_len,
            uint8_t * byte_array,
            size_t * array_len )
```

Decode base64 string to data.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_len*	Size of the base64 string in bytes.
out	*byte_array*	Decoded data will be returned here.
in,out	*array_len*	As input, the size of the byte_array buffer. As output, the length of the decoded data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.2 atcab_base64decode_()

```
ATCA_STATUS atcab_base64decode_ (
            const char * encoded,
            size_t encoded_size,
            uint8_t * data,
            size_t * data_size,
            const uint8_t * rules )
```

Decode base64 string to data with ruleset option.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_size*	Size of the base64 string in bytes.
out	*data*	Decoded data will be returned here.
in,out	*data_size*	As input, the size of the byte_array buffer. As output, the length of the decoded data.
in	*rules*	base64 ruleset to use

10.34.3.3 atcab_base64encode()

```
ATCA_STATUS atcab_base64encode (
            const uint8_t * byte_array,
            size_t array_len,
            char * encoded,
            size_t * encoded_len )
```

Encode data as base64 string.

Parameters

in	*byte_array*	Data to be encode in base64.
in	*array_len*	Size of byte_array in bytes.
in	*encoded*	Base64 output is returned here.
in,out	*encoded_len*	As input, the size of the encoded buffer. As output, the length of the encoded base64 character string.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.4 atcab_base64encode_()

```
ATCA_STATUS atcab_base64encode_ (
            const uint8_t * data,
            size_t data_size,
            char * encoded,
            size_t * encoded_size,
            const uint8_t * rules )
```

Encode data as base64 string with ruleset option.

Parameters

in	*data*	The input byte array that will be converted to base 64 encoded characters
in	*data_size*	The length of the byte array
in	*encoded*	The output converted to base 64 encoded characters.
in,out	*encoded_size*	Input: The size of the encoded buffer, Output: The length of the encoded base 64 character string
in	*rules*	ruleset to use during encoding

10.34.3.5 atcab_bin2hex()

```
ATCA_STATUS atcab_bin2hex (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size )
```

Convert a binary buffer to a hex string for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.6 atcab_bin2hex_()

```
ATCA_STATUS atcab_bin2hex_ (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size,
            bool is_pretty,
            bool is_space,
            bool is_upper )
```

Function that converts a binary buffer to a hex string suitable for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.
in	*is_pretty*	Indicates whether new lines should be added for pretty printing.
in	*is_space*	Convert the output hex with space between it.
in	*is_upper*	Convert the output hex to upper case.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.7 atcab_hex2bin()

```
ATCA_STATUS atcab_hex2bin (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size )
```

Function that converts a hex string to binary buffer.

Parameters

in	*hex*	Input buffer to convert
in	*hex_size*	Length of buffer to convert
out	*bin*	Buffer that receives binary
in,out	*bin_size*	As input, the size of the bin buffer. As output, the size of the bin data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.8 atcab_hex2bin_()

```
ATCA_STATUS atcab_hex2bin_ (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size,
            bool is_space )
```

10.34.3.9 atcab_memset_s()

```
int atcab_memset_s (
            void * dest,
            size_t destsz,
            int ch,
            size_t count )
```

Guaranteed to perform memory writes regardless of optimization level. Matches memset_s signature.

10.34.3.10 atcab_reversal()

```
ATCA_STATUS atcab_reversal (
            const uint8_t * bin,
            size_t bin_size,
            uint8_t * dest,
            size_t * dest_size )
```

To reverse the input data.

Parameters

in	*bin*	Input data to reverse.
in	*bin_size*	Size of data to reverse.
out	*dest*	Buffer to store reversed binary data.
in	*dest_size*	The size of the dest buffer.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.3.11 base64Char()

```
char base64Char (
            uint8_t id,
            const uint8_t * rules )
```

Returns the base 64 character of the given index.

Parameters

in	*id*	index to check
in	*rules*	base64 ruleset to use

Returns

the base 64 character of the given index

10.34.3.12 base64Index()

```
uint8_t base64Index (
            char c,
            const uint8_t * rules )
```

Returns the base 64 index of the given character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

the base 64 index of the given character

10.34.3.13 isAlpha()

```
bool isAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

10.34.3.14 isBase64()

```
bool isBase64 (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character or if this is space (A character can be included in a valid base 64 string).

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

10.34.3.15 isBase64Digit()

```
bool isBase64Digit (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

10.34.3.16 isBlankSpace()

```
bool isBlankSpace (
            char c )
```

Checks to see if a character is blank space.

Parameters

in	*c*	character to check

Returns

True if the character is blankspace

10.34.3.17 isDigit()

```
bool isDigit (
            char c )
```

Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))

Parameters

in	*c*	character to check

Returns

True if the character is a digit

10.34.3.18 isHex()

```
bool isHex (
            char c )
```

Returns true if this character is a valid hex character or if this is blankspace (The character can be included in a valid hexstring).

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

10.34.3.19 isHexAlpha()

```
bool isHexAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

10.34.3.20 isHexDigit()

```
bool isHexDigit (
            char c )
```

Returns true if this character is a valid hex character.

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

10.34.3.21 packHex()

```
ATCA_STATUS packHex (
            const char * ascii_hex,
            size_t ascii_hex_len,
            char * packed_hex,
            size_t * packed_len )
```

Remove spaces from a ASCII hex string.

Parameters

in	*ascii_hex*	Initial hex string to remove blankspace from
in	*ascii_hex_len*	Length of the initial hex string
in	*packed_hex*	Resulting hex string without blankspace
in,out	*packed_len*	In: Size to packed_hex buffer Out: Number of bytes in the packed hex string

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.34.4 Variable Documentation

10.34.4.1 atcab_b64rules_default

```
uint8_t atcab_b64rules_default[4] = { '+', '/', '=', 64 }
```

10.34.4.2 atcab_b64rules_mime

```
uint8_t atcab_b64rules_mime[4] = { '+', '/', '=', 76 }
```

10.34.4.3 atcab_b64rules_urlsafe

```
uint8_t atcab_b64rules_urlsafe[4] = { '-', '_', 0, 0 }
```

10.35 atca_helpers.h File Reference

Helpers to support the CryptoAuthLib Basic API methods.

```
#include "cryptoauthlib.h"
```

- uint8_t atcab_b64rules_default [4]
- uint8_t atcab_b64rules_mime [4]
- uint8_t atcab_b64rules_urlsafe [4]
- ATCA_STATUS atcab_printbin (uint8_t *binary, size_t bin_len, bool add_space)
- ATCA_STATUS atcab_bin2hex (const uint8_t *bin, size_t bin_size, char *hex, size_t *hex_size)

  Convert a binary buffer to a hex string for easy reading.
- ATCA_STATUS atcab_bin2hex_ (const uint8_t *bin, size_t bin_size, char *hex, size_t *hex_size, bool is_pretty, bool is_space, bool is_upper)

  Function that converts a binary buffer to a hex string suitable for easy reading.
- ATCA_STATUS atcab_hex2bin (const char *ascii_hex, size_t ascii_hex_len, uint8_t *binary, size_t *bin_len)

  Function that converts a hex string to binary buffer.
- ATCA_STATUS atcab_hex2bin_ (const char *hex, size_t hex_size, uint8_t *bin, size_t *bin_size, bool is_space)
- ATCA_STATUS atcab_printbin_sp (uint8_t *binary, size_t bin_len)
- ATCA_STATUS atcab_printbin_label (const char *label, uint8_t *binary, size_t bin_len)
- ATCA_STATUS packHex (const char *ascii_hex, size_t ascii_hex_len, char *packed_hex, size_t *packed_len)

  Remove spaces from a ASCII hex string.
- bool isDigit (char c)

  Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))

- bool isBlankSpace (char c)

    Checks to see if a character is blank space.
- bool isAlpha (char c)

    Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHexAlpha (char c)

    Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHex (char c)

    Returns true if this character is a valid hex character or if this is blankspace (The character can be included in a valid hexstring).
- bool isHexDigit (char c)

    Returns true if this character is a valid hex character.
- bool isBase64 (char c, const uint8_t ∗rules)

    Returns true if this character is a valid base 64 character or if this is space (A character can be included in a valid base 64 string).
- bool isBase64Digit (char c, const uint8_t ∗rules)

    Returns true if this character is a valid base 64 character.
- uint8_t base64Index (char c, const uint8_t ∗rules)

    Returns the base 64 index of the given character.
- char base64Char (uint8_t id, const uint8_t ∗rules)

    Returns the base 64 character of the given index.
- ATCA_STATUS atcab_base64decode_ (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_t ∗data_size, const uint8_t ∗rules)

    Decode base64 string to data with ruleset option.
- ATCA_STATUS atcab_base64decode (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_↩ t ∗data_size)

    Decode base64 string to data.
- ATCA_STATUS atcab_base64encode_ (const uint8_t ∗data, size_t data_size, char ∗encoded, size_↩ t ∗encoded_size, const uint8_t ∗rules)

    Encode data as base64 string with ruleset option.
- ATCA_STATUS atcab_base64encode (const uint8_t ∗data, size_t data_size, char ∗encoded, size_↩ t ∗encoded_size)

    Encode data as base64 string.
- ATCA_STATUS atcab_reversal (const uint8_t ∗bin, size_t bin_size, uint8_t ∗dest, size_t ∗dest_size)

    To reverse the input data.
- int atcab_memset_s (void ∗dest, size_t destsz, int ch, size_t count)

    Guaranteed to perform memory writes regardless of optimization level. Matches memset_s signature.

10.35.1 Detailed Description

Helpers to support the CryptoAuthLib Basic API methods.

Copyright

10.35.2 Function Documentation

10.35.2.1 atcab_base64decode()

```
ATCA_STATUS atcab_base64decode (
            const char * encoded,
            size_t encoded_len,
            uint8_t * byte_array,
            size_t * array_len )
```

Decode base64 string to data.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_len*	Size of the base64 string in bytes.
out	*byte_array*	Decoded data will be returned here.
in,out	*array_len*	As input, the size of the byte_array buffer. As output, the length of the decoded data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.2 atcab_base64decode_()

```
ATCA_STATUS atcab_base64decode_ (
            const char * encoded,
            size_t encoded_size,
            uint8_t * data,
            size_t * data_size,
            const uint8_t * rules )
```

Decode base64 string to data with ruleset option.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_size*	Size of the base64 string in bytes.
out	*data*	Decoded data will be returned here.
in,out	*data_size*	As input, the size of the byte_array buffer. As output, the length of the decoded data.
in	*rules*	base64 ruleset to use

10.35.2.3 atcab_base64encode()

```
ATCA_STATUS atcab_base64encode (
            const uint8_t * byte_array,
            size_t array_len,
```

```
            char * encoded,
            size_t * encoded_len )
```

Encode data as base64 string.

Parameters

in	*byte_array*	Data to be encode in base64.
in	*array_len*	Size of byte_array in bytes.
in	*encoded*	Base64 output is returned here.
in,out	*encoded_len*	As input, the size of the encoded buffer. As output, the length of the encoded base64 character string.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.4 atcab_base64encode_()

```
ATCA_STATUS atcab_base64encode_ (
            const uint8_t * data,
            size_t data_size,
            char * encoded,
            size_t * encoded_size,
            const uint8_t * rules )
```

Encode data as base64 string with ruleset option.

Parameters

in	*data*	The input byte array that will be converted to base 64 encoded characters
in	*data_size*	The length of the byte array
in	*encoded*	The output converted to base 64 encoded characters.
in,out	*encoded_size*	Input: The size of the encoded buffer, Output: The length of the encoded base 64 character string
in	*rules*	ruleset to use during encoding

10.35.2.5 atcab_bin2hex()

```
ATCA_STATUS atcab_bin2hex (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size )
```

Convert a binary buffer to a hex string for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.6 atcab_bin2hex_()

```
ATCA_STATUS atcab_bin2hex_ (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size,
            bool is_pretty,
            bool is_space,
            bool is_upper )
```

Function that converts a binary buffer to a hex string suitable for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.
in	*is_pretty*	Indicates whether new lines should be added for pretty printing.
in	*is_space*	Convert the output hex with space between it.
in	*is_upper*	Convert the output hex to upper case.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.7 atcab_hex2bin()

```
ATCA_STATUS atcab_hex2bin (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size )
```

Function that converts a hex string to binary buffer.

Parameters

in	*hex*	Input buffer to convert
in	*hex_size*	Length of buffer to convert
out	*bin*	Buffer that receives binary
in,out	*bin_size*	As input, the size of the bin buffer. As output, the size of the bin data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.8 atcab_hex2bin_()

```
ATCA_STATUS atcab_hex2bin_ (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size,
            bool is_space )
```

10.35.2.9 atcab_memset_s()

```
int atcab_memset_s (
            void * dest,
            size_t destsz,
            int ch,
            size_t count )
```

Guaranteed to perform memory writes regardless of optimization level. Matches memset_s signature.

10.35.2.10 atcab_printbin_label()

```
ATCA_STATUS atcab_printbin_label (
            const char * label,
            uint8_t * binary,
            size_t bin_len )
```

10.35.2.11 atcab_printbin_sp()

```
ATCA_STATUS atcab_printbin_sp (
            uint8_t * binary,
            size_t bin_len )
```

10.35.2.12 atcab_reversal()

```
ATCA_STATUS atcab_reversal (
            const uint8_t * bin,
            size_t bin_size,
            uint8_t * dest,
            size_t * dest_size )
```

To reverse the input data.

Parameters

in	*bin*	Input data to reverse.
in	*bin_size*	Size of data to reverse.
out	*dest*	Buffer to store reversed binary data.
in	*dest_size*	The size of the dest buffer.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.2.13 base64Char()

```
char base64Char (
            uint8_t id,
            const uint8_t * rules )
```

Returns the base 64 character of the given index.

Parameters

in	*id*	index to check
in	*rules*	base64 ruleset to use

Returns

the base 64 character of the given index

10.35.2.14 base64Index()

```
uint8_t base64Index (
            char c,
            const uint8_t * rules )
```

Returns the base 64 index of the given character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

the base 64 index of the given character

10.35.2.15 isAlpha()

```
bool isAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

10.35.2.16 isBase64()

```
bool isBase64 (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character or if this is space (A character can be included in a valid base 64 string).

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

10.35.2.17 isBase64Digit()

```
bool isBase64Digit (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

10.35.2.18 isBlankSpace()

```
bool isBlankSpace (
            char c )
```

Checks to see if a character is blank space.

Parameters

in	*c*	character to check

Returns

True if the character is blankspace

10.35.2.19 isDigit()

```
bool isDigit (
            char c )
```

Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))

Parameters

in	*c*	character to check

Returns

True if the character is a digit

10.35.2.20 isHex()

```
bool isHex (
            char c )
```

Returns true if this character is a valid hex character or if this is blankspace (The character can be included in a valid hexstring).

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

10.35.2.21 isHexAlpha()

```
bool isHexAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

10.35.2.22 isHexDigit()

```
bool isHexDigit (
            char c )
```

Returns true if this character is a valid hex character.

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

10.35.2.23 packHex()

```
ATCA_STATUS packHex (
            const char * ascii_hex,
            size_t ascii_hex_len,
            char * packed_hex,
            size_t * packed_len )
```

Remove spaces from a ASCII hex string.

Parameters

in	*ascii_hex*	Initial hex string to remove blankspace from
in	*ascii_hex_len*	Length of the initial hex string
in	*packed_hex*	Resulting hex string without blankspace
in,out	*packed_len*	In: Size to packed_hex buffer Out: Number of bytes in the packed hex string

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.35.3 Variable Documentation

10.35.3.1 atcab_b64rules_default

```
uint8_t atcab_b64rules_default[4] [extern]
```

10.35.3.2 atcab_b64rules_mime

```
uint8_t atcab_b64rules_mime[4] [extern]
```

10.35.3.3 atcab_b64rules_urlsafe

```
uint8_t atcab_b64rules_urlsafe[4] [extern]
```

10.36 atca_host.c File Reference

Host side methods to support CryptoAuth computations.

```
#include "atca_host.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Functions

- uint8_t ∗ atcah_include_data (struct atca_include_data_in_out ∗param)
  This function copies otp and sn data into a command buffer.
- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out ∗param)
  This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out ∗param)
  Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608 are the only ones that support this operation.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t ∗param)
  Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t ∗param)
  Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t ∗param)
  Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out ∗param)
  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out ∗param)
  This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out ∗param)
  This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out ∗param)
  This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out ∗param)
  This function generates mac with session key with a plain text.
- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out ∗param)
  This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out ∗param)
  This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out ∗param)
  This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out ∗param)
  This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out ∗param)
  This function decrypts 32-byte encrypted data received with the Read command.
- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t ∗message, uint8_t ∗digest)
  This function creates a SHA256 digest on a little-endian system.

- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out ∗param)

  Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param, const uint8_t ∗config)

  Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param)

  Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter_value, uint8_t ∗counter_match_value)

  Builds the counter match value that needs to be stored in a slot.
- ATCA_STATUS atcah_ecc204_write_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the ECC204 Write command.
- ATCA_STATUS atcah_gen_session_key (struct atca_session_key_in_out ∗param)

  This function calculates the session key for the ECC204.

10.36.1 Detailed Description

Host side methods to support CryptoAuth computations.

Copyright

10.37 atca_host.h File Reference

Definitions and Prototypes for ATCA Utility Functions.

```
#include <stdint.h>
#include "cryptoauthlib.h"
#include "calib/calib_basic.h"
```

Data Structures

- struct atca_temp_key

  Structure to hold TempKey fields.
- struct atca_include_data_in_out

  Input / output parameters for function atca_include_data().
- struct atca_nonce_in_out

  Input/output parameters for function atca_nonce().
- struct atca_io_decrypt_in_out
- struct atca_verify_mac
- struct atca_secureboot_enc_in_out
- struct atca_secureboot_mac_in_out
- struct atca_mac_in_out

  Input/output parameters for function atca_mac().
- struct atca_hmac_in_out

  Input/output parameters for function atca_hmac().

- struct atca_gen_dig_in_out

  Input/output parameters for function atcah_gen_dig().
- struct atca_write_mac_in_out

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- struct atca_derive_key_in_out

  Input/output parameters for function atcah_derive_key().
- struct atca_derive_key_mac_in_out

  Input/output parameters for function atcah_derive_key_mac().
- struct atca_decrypt_in_out

  Input/output parameters for function atca_decrypt().
- struct atca_check_mac_in_out

  Input/output parameters for function atcah_check_mac().
- struct atca_verify_in_out

  Input/output parameters for function atcah_verify().
- struct atca_gen_key_in_out

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- struct atca_sign_internal_in_out

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.
- struct atca_session_key_in_out

  Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

Macros

Definitions for ATECC Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define ATCA_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.
- #define ATCA_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}
- #define ATCA_MSG_SIZE_HMAC (88)
- #define ATCA_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.
- #define ATCA_MSG_SIZE_ENCRYPT_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_SESSION_KEY (96)

  TransportKey{32} || 0x15{1} || 0x00{1} || KeyId{2} || SN8{1} || SN0_1{2} || 0{25} || Nonce{32}.
- #define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.
- #define ATCA_COMMAND_HEADER_SIZE (4)
- #define ATCA_GENDIG_ZEROS_SIZE (25)
- #define ATCA_WRITE_MAC_ZEROS_SIZE (25)
- #define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
- #define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
- #define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
- #define ATCA_HMAC_BLOCK_SIZE (64)
- #define ENCRYPTION_KEY_SIZE (64)

Default Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define ATCA_SN_0_DEF (0x01)
- #define ATCA_SN_1_DEF (0x23)
- #define ATCA_SN_8_DEF (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)
  mode mask for MAC command when using TempKey

Typedefs

- typedef struct atca_temp_key atca_temp_key_t
  Structure to hold TempKey fields.
- typedef struct atca_nonce_in_out atca_nonce_in_out_t
- typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
- typedef struct atca_verify_mac atca_verify_mac_in_out_t
- typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
- typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
- typedef struct atca_mac_in_out atca_mac_in_out_t
- typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t
  Input/output parameters for function atcah_gen_dig().
- typedef struct atca_write_mac_in_out atca_write_mac_in_out_t
  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- typedef struct atca_check_mac_in_out atca_check_mac_in_out_t
  Input/output parameters for function atcah_check_mac().
- typedef struct atca_verify_in_out atca_verify_in_out_t
- typedef struct atca_gen_key_in_out atca_gen_key_in_out_t
  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t
  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.
- typedef struct atca_session_key_in_out atca_session_key_in_out_t
  Input/Output paramters for calculating the session key by the nonce command. Used with the atcah_gen_session_key() function.

Functions

- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out ∗param)
  This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out ∗param)
  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out ∗param)
  This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out ∗param)
  This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out ∗param)
  This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out ∗param)

This function generates mac with session key with a plain text.

- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out ∗param)

  This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out ∗param)

  This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data received with the Read command.
- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.
- uint8_t ∗ atcah_include_data (struct atca_include_data_in_out ∗param)

  This function copies otp and sn data into a command buffer.
- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out ∗param)

  Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param, const uint8_t ∗config)

  Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param)

  Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t ∗param)

  Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t ∗param)

  Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t ∗param)

  Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter, uint8_t ∗counter_match)

  Builds the counter match value that needs to be stored in a slot.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out ∗param)

  Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608 are the only ones that support this operation.
- ATCA_STATUS atcah_ecc204_write_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the ECC204 Write command.
- ATCA_STATUS atcah_gen_session_key (atca_session_key_in_out_t ∗param)

  This function calculates the session key for the ECC204.

10.37.1 Detailed Description

Definitions and Prototypes for ATCA Utility Functions.

Copyright

10.38 atca_iface.c File Reference

Microchip CryptoAuthLib hardware interface object.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS initATCAIface (ATCAIfaceCfg *cfg, ATCAIface ca_iface)

    Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg *cfg)

    Constructor for ATCAIface objects.
- ATCA_STATUS atinit (ATCAIface ca_iface)

    Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t address, uint8_t *txdata, int txlength)

    Sends the data to the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t word_address, uint8_t *rxdata, uint16_t *rxlength)

    Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atcontrol (ATCAIface ca_iface, uint8_t option, void *param, size_t paramlen)

    Perform control operations with the underlying hal driver.
- ATCA_STATUS atwake (ATCAIface ca_iface)

    Wakes up the device by calling intermediate HAL wrapper function. The atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

    Puts the device into idle state by calling intermediate HAL wrapper function. The atcab_idle() function should be used instead.
- ATCA_STATUS atsleep (ATCAIface ca_iface)

    Puts the device into sleep state by calling intermediate HAL wrapper function. The atcab_sleep() function should be used instead.
- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

    Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

    Returns the HAL data pointer for the device.
- bool atca_iface_is_kit (ATCAIface ca_iface)

    Check if the given interface is configured as a "kit protocol" one where transactions are atomic.
- int atca_iface_get_retries (ATCAIface ca_iface)

    Retrive the number of retries for a configured interface.
- uint16_t atca_iface_get_wake_delay (ATCAIface ca_iface)

    Retrive the wake/retry delay for a configured interface/device.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

    Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface *ca_iface)

    Instruct the HAL driver to release any resources associated with this interface, then delete the object.

10.38.1 Detailed Description

Microchip CryptoAuthLib hardware interface object.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.39 atca_iface.h File Reference

Microchip Crypto Auth hardware interface object.

```
#include <stdint.h>
#include <stddef.h>
#include "atca_config.h"
#include "atca_devtypes.h"
#include "atca_status.h"
```

Data Structures

- struct ATCAIfaceCfg
- struct ATCAHAL_t

    HAL Driver Structure.
- struct atca_iface

    atca_iface is the context structure for a configured interface

Typedefs

- typedef struct atca_iface * ATCAIface
- typedef struct atca_iface atca_iface_t

    atca_iface is the context structure for a configured interface

Enumerations

- enum ATCAIfaceType {
  ATCA_I2C_IFACE = 0, ATCA_SWI_IFACE = 1, ATCA_UART_IFACE = 2, ATCA_SPI_IFACE = 3,
  ATCA_HID_IFACE = 4, ATCA_KIT_IFACE = 5, ATCA_CUSTOM_IFACE = 6, ATCA_I2C_GPIO_IFACE = 7,
  ATCA_SWI_GPIO_IFACE = 8, ATCA_SPI_GPIO_IFACE = 9, ATCA_UNKNOWN_IFACE = 0xFE }
- enum ATCAKitType {
  ATCA_KIT_AUTO_IFACE, ATCA_KIT_I2C_IFACE, ATCA_KIT_SWI_IFACE, ATCA_KIT_SPI_IFACE,
  ATCA_KIT_UNKNOWN_IFACE }

Functions

- ATCA_STATUS initATCAIface (ATCAIfaceCfg *cfg, ATCAIface ca_iface)

    Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg *cfg)

    Constructor for ATCAIface objects.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

    Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface *ca_iface)

    Instruct the HAL driver to release any resources associated with this interface, then delete the object.
- ATCA_STATUS atinit (ATCAIface ca_iface)

    Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t address, uint8_t *txdata, int txlength)

Sends the data to the device by calling intermediate HAL wrapper function.

- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t word_address, uint8_t *rxdata, uint16_t *rxlength)

    Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atcontrol (ATCAIface ca_iface, uint8_t option, void *param, size_t paramlen)

    Perform control operations with the underlying hal driver.
- ATCA_STATUS atwake (ATCAIface ca_iface)

    Wakes up the device by calling intermediate HAL wrapper function. The atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

    Puts the device into idle state by calling intermediate HAL wrapper function. The atcab_idle() function should be used instead.
- ATCA_STATUS atsleep (ATCAIface ca_iface)

    Puts the device into sleep state by calling intermediate HAL wrapper function. The atcab_sleep() function should be used instead.
- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

    Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

    Returns the HAL data pointer for the device.
- bool atca_iface_is_kit (ATCAIface ca_iface)

    Check if the given interface is configured as a "kit protocol" one where transactions are atomic.
- int atca_iface_get_retries (ATCAIface ca_iface)

    Retrive the number of retries for a configured interface.
- uint16_t atca_iface_get_wake_delay (ATCAIface ca_iface)

    Retrive the wake/retry delay for a configured interface/device.

10.39.1 Detailed Description

Microchip Crypto Auth hardware interface object.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.40 atca_jwt.c File Reference

Utilities to create and verify a JSON Web Token (JWT)

```
#include "cryptoauthlib.h"
#include "atca_helpers.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "jwt/atca_jwt.h"
#include <stdio.h>
```

Functions

- void atca_jwt_check_payload_start (atca_jwt_t ∗jwt)

  Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_init (atca_jwt_t ∗jwt, char ∗buf, uint16_t buflen)

  Initialize a JWT structure.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t ∗jwt, uint16_t key_id)

  Close the claims of a token, encode them, then sign the result.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t ∗jwt, const char ∗claim, const char ∗value)

  Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t ∗jwt, const char ∗claim, int32_t value)

  Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_verify (const char ∗buf, uint16_t buflen, const uint8_t ∗pubkey)

  Verifies the signature of a jwt using the provided public key.

10.40.1 Detailed Description

Utilities to create and verify a JSON Web Token (JWT)

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.41 atca_jwt.h File Reference

Utilities to create and verify a JSON Web Token (JWT)

```
#include "cryptoauthlib.h"
```

Data Structures

- struct atca_jwt_t

  Structure to hold metadata information about the jwt being built.

Functions

- ATCA_STATUS atca_jwt_init (atca_jwt_t ∗jwt, char ∗buf, uint16_t buflen)

  Initialize a JWT structure.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t ∗jwt, const char ∗claim, const char ∗value)

  Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t ∗jwt, const char ∗claim, int32_t value)

  Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t ∗jwt, uint16_t key_id)

  Close the claims of a token, encode them, then sign the result.
- void atca_jwt_check_payload_start (atca_jwt_t ∗jwt)

  Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_verify (const char ∗buf, uint16_t buflen, const uint8_t ∗pubkey)

  Verifies the signature of a jwt using the provided public key.

10.41.1 Detailed Description

Utilities to create and verify a JSON Web Token (JWT)

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.42 atca_mbedtls_ecdh.c File Reference

```
#include "mbedtls/config.h"
```

10.43 atca_mbedtls_ecdsa.c File Reference

```
#include "mbedtls/config.h"
```

10.44 atca_mbedtls_wrap.c File Reference

Wrapper functions to replace cryptoauthlib software crypto functions with the mbedTLS equivalent.

```
#include "mbedtls/config.h"
#include <stdlib.h>
#include "mbedtls/cmac.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/pk.h"
#include "mbedtls/ecdh.h"
#include "mbedtls/ecp.h"
#include "mbedtls/entropy.h"
#include "mbedtls/bignum.h"
#include "mbedtls/x509_crt.h"
#include "cryptoauthlib.h"
#include "atca_mbedtls_wrap.h"
#include "atca_mbedtls_patch.h"
#include "crypto/atca_crypto_sw.h"
#include "atcacert/atcacert_client.h"
#include "atcacert/atcacert_def.h"
#include "mbedtls/pk_internal.h"
#include "atcacert/atcacert_der.h"
```

Macros

- #define mbedtls_calloc calloc
- #define mbedtls_free free

Functions

- int atcac_sw_random (uint8_t ∗data, size_t data_size)

  Return Random Bytes.
- ATCA_STATUS atcac_aes_gcm_aad_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗aad, const size_t aad_len)

  Update the GCM context with additional authentication data (AAD)
- ATCA_STATUS atcac_aes_gcm_encrypt_start (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len, const uint8_t ∗iv, const uint8_t iv_len)

  Initialize an AES-GCM context.
- ATCA_STATUS atcac_aes_gcm_encrypt_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗plaintext, const size_t pt_len, uint8_t ∗ciphertext, size_t ∗ct_len)

  Encrypt a data using the initialized context.
- ATCA_STATUS atcac_aes_gcm_encrypt_finish (atcac_aes_gcm_ctx ∗ctx, uint8_t ∗tag, size_t tag_len)

  Get the AES-GCM tag and free the context.
- ATCA_STATUS atcac_aes_gcm_decrypt_start (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len, const uint8_t ∗iv, const uint8_t iv_len)

  Initialize an AES-GCM context for decryption.
- ATCA_STATUS atcac_aes_gcm_decrypt_update (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗ciphertext, const size_t ct_len, uint8_t ∗plaintext, size_t ∗pt_len)

  Decrypt ciphertext using the initialized context.
- ATCA_STATUS atcac_aes_gcm_decrypt_finish (atcac_aes_gcm_ctx ∗ctx, const uint8_t ∗tag, size_t tag_len, bool ∗is_verified)

  Compare the AES-GCM tag and free the context.
- int atcac_sw_sha1_init (atcac_sha1_ctx ∗ctx)

  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Add data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx ∗ctx, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Complete the SHA1 hash in software and return the digest.
- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx ∗ctx)

  Initialize context for performing SHA256 hash in software.
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Add data to a SHA256 hash.
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx ∗ctx, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  Complete the SHA256 hash in software and return the digest.
- ATCA_STATUS atcac_aes_cmac_init (atcac_aes_cmac_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len)

  Initialize context for performing CMAC in software.
- ATCA_STATUS atcac_aes_cmac_update (atcac_aes_cmac_ctx ∗ctx, const uint8_t ∗data, const size_t data_size)

  Update CMAC context with input data.
- ATCA_STATUS atcac_aes_cmac_finish (atcac_aes_cmac_ctx ∗ctx, uint8_t ∗cmac, size_t ∗cmac_size)

  Finish CMAC calculation and clear the CMAC context.
- ATCA_STATUS atcac_sha256_hmac_init (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗key, const uint8_t key_len)

  Initialize context for performing HMAC (sha256) in software.
- ATCA_STATUS atcac_sha256_hmac_update (atcac_hmac_sha256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Update HMAC context with input data.
- ATCA_STATUS atcac_sha256_hmac_finish (atcac_hmac_sha256_ctx ∗ctx, uint8_t ∗digest, size_t ∗digest_len)

  Finish CMAC calculation and clear the HMAC context.

- ATCA_STATUS atcac_pk_init (atcac_pk_ctx *ctx, uint8_t *buf, size_t buflen, uint8_t key_type, bool pubkey)

  Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_init_pem (atcac_pk_ctx *ctx, uint8_t *buf, size_t buflen, bool pubkey)

  Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_free (atcac_pk_ctx *ctx)

  Free a public/private key structure.
- ATCA_STATUS atcac_pk_public (atcac_pk_ctx *ctx, uint8_t *buf, size_t *buflen)

  Get the public key from the context.
- ATCA_STATUS atcac_pk_sign (atcac_pk_ctx *ctx, uint8_t *digest, size_t dig_len, uint8_t *signature, size_t *sig_len)

  Perform a signature with the private key in the context.
- ATCA_STATUS atcac_pk_verify (atcac_pk_ctx *ctx, uint8_t *digest, size_t dig_len, uint8_t *signature, size_t sig_len)

  Perform a verify using the public key in the provided context.
- ATCA_STATUS atcac_pk_derive (atcac_pk_ctx *private_ctx, atcac_pk_ctx *public_ctx, uint8_t *buf, size_t *buflen)

  Execute the key agreement protocol for the provided keys (if they can)
- int atca_mbedtls_pk_init_ext (ATCADevice device, mbedtls_pk_context *pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_pk_init (mbedtls_pk_context *pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (mbedtls_x509_crt *cert, const atcacert_def_t *cert_def)

  Rebuild a certificate from an atcacert_def_t structure, and then add it to an mbedtls cert chain.

Variables

- const mbedtls_pk_info_t atca_mbedtls_eckey_info

10.44.1 Detailed Description

Wrapper functions to replace cryptoauthlib software crypto functions with the mbedTLS equivalent.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.44.2 Macro Definition Documentation

10.44.2.1 mbedtls_calloc

```
#define mbedtls_calloc calloc
```

10.44.2.2 mbedtls_free

```
#define mbedtls_free free
```

10.44.3 Function Documentation

10.44.3.1 atca_mbedtls_cert_add()

```
int atca_mbedtls_cert_add (
            mbedtls_x509_crt * cert,
            const atcacert_def_t * cert_def )
```

Rebuild a certificate from an atcacert_def_t structure, and then add it to an mbedtls cert chain.

Parameters

in,out	*cert*	mbedtls cert chain. Must have already been initialized
in	*cert_def*	Certificate definition that will be rebuilt and added

Returns

0 on success, otherwise an error code.

10.44.3.2 atcac_aes_cmac_finish()

```
ATCA_STATUS atcac_aes_cmac_finish (
            atcac_aes_cmac_ctx * ctx,
            uint8_t * cmac,
            size_t * cmac_size )
```

Finish CMAC calculation and clear the CMAC context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
out	*cmac*	cmac value
in,out	*cmac_size*	length of cmac

10.44.3.3 atcac_aes_cmac_init()

```
ATCA_STATUS atcac_aes_cmac_init (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len )
```

Initialize context for performing CMAC in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*key*	key value to use
in	*key_len*	length of the key

10.44.3.4 atcac_aes_cmac_update()

```
ATCA_STATUS atcac_aes_cmac_update (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * data,
            const size_t data_size )
```

Update CMAC context with input data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*data*	input data
in	*data_size*	length of input data

10.44.3.5 atcac_aes_gcm_aad_update()

```
ATCA_STATUS atcac_aes_gcm_aad_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * aad,
            const size_t aad_len )
```

Update the GCM context with additional authentication data (AAD)

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*aad*	Additional Authentication Data
in	*aad_len*	Length of AAD

10.44.3.6 atcac_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_decrypt_finish (
             atcac_aes_gcm_ctx * ctx,
             const uint8_t * tag,
             size_t tag_len,
             bool * is_verified )
```

Compare the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*tag*	GCM Tag to Verify
in	*tag_len*	Length of the GCM tag
out	*is_verified*	Tag verified as matching

10.44.3.7 atcac_aes_gcm_decrypt_start()

```
ATCA_STATUS atcac_aes_gcm_decrypt_start (
             atcac_aes_gcm_ctx * ctx,
             const uint8_t * key,
             const uint8_t key_len,
             const uint8_t * iv,
             const uint8_t iv_len )
```

Initialize an AES-GCM context for decryption.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.44.3.8 atcac_aes_gcm_decrypt_update()

```
ATCA_STATUS atcac_aes_gcm_decrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * ciphertext,
            const size_t ct_len,
            uint8_t * plaintext,
            size_t * pt_len )
```

Decrypt ciphertext using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*ciphertext*	Ciphertext to decrypt
in	*ct_len*	Length of the ciphertext
out	*plaintext*	Resulting decrypted plaintext
in,out	*pt_len*	Length of the plaintext buffer

10.44.3.9 atcac_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_encrypt_finish (
            atcac_aes_gcm_ctx * ctx,
            uint8_t * tag,
            size_t tag_len )
```

Get the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
out	*tag*	GCM Tag Result
in	*tag_len*	Length of the GCM tag

10.44.3.10 atcac_aes_gcm_encrypt_start()

```
ATCA_STATUS atcac_aes_gcm_encrypt_start (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len,
            const uint8_t * iv,
            const uint8_t iv_len )
```

Initialize an AES-GCM context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.44.3.11 atcac_aes_gcm_encrypt_update()

```
ATCA_STATUS atcac_aes_gcm_encrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * plaintext,
            const size_t pt_len,
            uint8_t * ciphertext,
            size_t * ct_len )
```

Encrypt a data using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*plaintext*	Input buffer to encrypt
in	*pt_len*	Length of the input
out	*ciphertext*	Output buffer
in,out	*ct_len*	Length of the ciphertext buffer

10.44.3.12 atcac_pk_derive()

```
ATCA_STATUS atcac_pk_derive (
            atcac_pk_ctx * private_ctx,
            atcac_pk_ctx * public_ctx,
            uint8_t * buf,
            size_t * buflen )
```

Execute the key agreement protocol for the provided keys (if they can)

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.44.3.13 atcac_pk_free()

```
ATCA_STATUS atcac_pk_free (
            atcac_pk_ctx * ctx )
```

Free a public/private key structure.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context

10.44.3.14 atcac_pk_init()

```
ATCA_STATUS atcac_pk_init (
            atcac_pk_ctx * ctx,
```

```
            uint8_t * buf,
            size_t buflen,
            uint8_t key_type,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.44.3.15 atcac_pk_init_pem()

```
ATCA_STATUS atcac_pk_init_pem (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t buflen,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.44.3.16 atcac_pk_public()

```
ATCA_STATUS atcac_pk_public (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t * buflen )
```

Get the public key from the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.44.3.17 atcac_pk_sign()

```
ATCA_STATUS atcac_pk_sign (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t * sig_len )
```

Perform a signature with the private key in the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.44.3.18 atcac_pk_verify()

```
ATCA_STATUS atcac_pk_verify (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t sig_len )
```

Perform a verify using the public key in the provided context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.44.3.19 atcac_sw_sha1_finish()

```
int atcac_sw_sha1_finish (
            atcac_sha1_ctx * ctx,
            uint8_t digest[ATCA_SHA1_DIGEST_SIZE] )
```

Complete the SHA1 hash in software and return the digest.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
out	*digest*	output buffer (20 bytes)

10.44.3.20 atcac_sw_sha2_256_finish()

```
int atcac_sw_sha2_256_finish (
            atcac_sha2_256_ctx * ctx,
            uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE] )
```

Complete the SHA256 hash in software and return the digest.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
out	*digest*	output buffer (32 bytes)

10.44.4 Variable Documentation

10.44.4.1 atca_mbedtls_eckey_info

```
const mbedtls_pk_info_t atca_mbedtls_eckey_info
```

Initial value:

```
= {
    MBEDTLS_PK_ECKEY,
    "EC",
    atca_mbedtls_eckey_get_bitlen,
    atca_mbedtls_eckey_can_do,
    atca_mbedtls_eckey_verify,
    atca_mbedtls_eckey_sign,
    NULL,
    NULL,
    atca_mbedtls_eckey_check_pair,
    atca_mbedtls_eckey_alloc,
    atca_mbedtls_eckey_free,
    atca_mbedtls_eckey_debug,
}
```

10.45 atca_mbedtls_wrap.h File Reference

Data Structures

- struct atca_mbedtls_eckey_s

Typedefs

- typedef struct atca_mbedtls_eckey_s atca_mbedtls_eckey_t

Functions

- int atca_mbedtls_ecdsa_sign (const mbedtls_mpi *d, mbedtls_mpi *r, mbedtls_mpi *s, const unsigned char *buf, size_t buf_len)
- int atca_mbedtls_pk_init_ext (ATCADevice device, struct mbedtls_pk_context *pkey, const uint16_t slotid)

    Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_pk_init (struct mbedtls_pk_context *pkey, const uint16_t slotid)

    Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (struct mbedtls_x509_crt *cert, const struct atcacert_def_s *cert_def)
- int atca_mbedtls_ecdh_slot_cb (void)

    ECDH Callback to obtain the "slot" used in ECDH operations from the application.
- int atca_mbedtls_ecdh_ioprot_cb (uint8_t secret[32])

    ECDH Callback to obtain the IO Protection secret from the application.

10.46 atca_openssl_interface.c File Reference

Crypto abstraction functions for external host side cryptography.

```
#include "atca_config.h"
#include "atca_status.h"
#include "crypto/atca_crypto_sw.h"
#include <openssl/bn.h>
#include <openssl/bio.h>
#include <openssl/cmac.h>
#include <openssl/ec.h>
#include <openssl/evp.h>
#include <openssl/hmac.h>
#include <openssl/pem.h>
```

Functions

- ATCA_STATUS atcac_aes_gcm_aad_update (atcac_aes_gcm_ctx *ctx, const uint8_t *aad, const size_t aad_len)

    Update the GCM context with additional authentication data (AAD)
- ATCA_STATUS atcac_aes_gcm_encrypt_start (atcac_aes_gcm_ctx *ctx, const uint8_t *key, const uint8_t key_len, const uint8_t *iv, const uint8_t iv_len)

    Initialize an AES-GCM context.
- ATCA_STATUS atcac_aes_gcm_encrypt_update (atcac_aes_gcm_ctx *ctx, const uint8_t *plaintext, const size_t pt_len, uint8_t *ciphertext, size_t *ct_len)

    Encrypt a data using the initialized context.
- ATCA_STATUS atcac_aes_gcm_encrypt_finish (atcac_aes_gcm_ctx *ctx, uint8_t *tag, size_t tag_len)

    Get the AES-GCM tag and free the context.
- ATCA_STATUS atcac_aes_gcm_decrypt_start (atcac_aes_gcm_ctx *ctx, const uint8_t *key, const uint8_t key_len, const uint8_t *iv, const uint8_t iv_len)

    Initialize an AES-GCM context for decryption.

- ATCA_STATUS atcac_aes_gcm_decrypt_update (atcac_aes_gcm_ctx *ctx, const uint8_t *ciphertext, const size_t ct_len, uint8_t *plaintext, size_t *pt_len)

  Decrypt ciphertext using the initialized context.
- ATCA_STATUS atcac_aes_gcm_decrypt_finish (atcac_aes_gcm_ctx *ctx, const uint8_t *tag, size_t tag_len, bool *is_verified)

  Compare the AES-GCM tag and free the context.
- int atcac_sw_sha1_init (atcac_sha1_ctx *ctx)

  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx *ctx, const uint8_t *data, size_t data_size)

  Add data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx *ctx, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Complete the SHA1 hash in software and return the digest.
- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx *ctx)

  Initialize context for performing SHA256 hash in software.
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx *ctx, const uint8_t *data, size_t data_size)

  Add data to a SHA256 hash.
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx *ctx, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  Complete the SHA256 hash in software and return the digest.
- ATCA_STATUS atcac_aes_cmac_init (atcac_aes_cmac_ctx *ctx, const uint8_t *key, const uint8_t key_len)

  Initialize context for performing CMAC in software.
- ATCA_STATUS atcac_aes_cmac_update (atcac_aes_cmac_ctx *ctx, const uint8_t *data, const size_t data_size)

  Update CMAC context with input data.
- ATCA_STATUS atcac_aes_cmac_finish (atcac_aes_cmac_ctx *ctx, uint8_t *cmac, size_t *cmac_size)

  Finish CMAC calculation and clear the CMAC context.
- ATCA_STATUS atcac_sha256_hmac_init (atcac_hmac_sha256_ctx *ctx, const uint8_t *key, const uint8_t key_len)

  Initialize context for performing HMAC (sha256) in software.
- ATCA_STATUS atcac_sha256_hmac_update (atcac_hmac_sha256_ctx *ctx, const uint8_t *data, size_t data_size)

  Update HMAC context with input data.
- ATCA_STATUS atcac_sha256_hmac_finish (atcac_hmac_sha256_ctx *ctx, uint8_t *digest, size_t *digest_len)

  Finish CMAC calculation and clear the HMAC context.
- ATCA_STATUS atcac_pk_init (atcac_pk_ctx *ctx, uint8_t *buf, size_t buflen, uint8_t key_type, bool pubkey)

  Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_init_pem (atcac_pk_ctx *ctx, uint8_t *buf, size_t buflen, bool pubkey)

  Set up a public/private key structure for use in asymmetric cryptographic functions.
- ATCA_STATUS atcac_pk_free (atcac_pk_ctx *ctx)

  Free a public/private key structure.
- ATCA_STATUS atcac_pk_public (atcac_pk_ctx *ctx, uint8_t *buf, size_t *buflen)

  Get the public key from the context.
- ATCA_STATUS atcac_pk_sign (atcac_pk_ctx *ctx, uint8_t *digest, size_t dig_len, uint8_t *signature, size_t *sig_len)

  Perform a signature with the private key in the context.
- ATCA_STATUS atcac_pk_verify (atcac_pk_ctx *ctx, uint8_t *digest, size_t dig_len, uint8_t *signature, size_t sig_len)

  Perform a verify using the public key in the provided context.
- ATCA_STATUS atcac_pk_derive (atcac_pk_ctx *private_ctx, atcac_pk_ctx *public_ctx, uint8_t *buf, size_t *buflen)

  Execute the key agreement protocol for the provided keys (if they can)

10.46.1 Detailed Description

Crypto abstraction functions for external host side cryptography.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.46.2 Function Documentation

10.46.2.1 atcac_aes_cmac_finish()

```
ATCA_STATUS atcac_aes_cmac_finish (
            atcac_aes_cmac_ctx * ctx,
            uint8_t * cmac,
            size_t * cmac_size )
```

Finish CMAC calculation and clear the CMAC context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
out	*cmac*	cmac value
in,out	*cmac_size*	length of cmac

10.46.2.2 atcac_aes_cmac_init()

```
ATCA_STATUS atcac_aes_cmac_init (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len )
```

Initialize context for performing CMAC in software.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*key*	key value to use
in	*key_len*	length of the key

10.46.2.3 atcac_aes_cmac_update()

```
ATCA_STATUS atcac_aes_cmac_update (
            atcac_aes_cmac_ctx * ctx,
            const uint8_t * data,
            const size_t data_size )
```

Update CMAC context with input data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a aes-cmac context
in	*data*	input data
in	*data_size*	length of input data

10.46.2.4 atcac_aes_gcm_aad_update()

```
ATCA_STATUS atcac_aes_gcm_aad_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * aad,
            const size_t aad_len )
```

Update the GCM context with additional authentication data (AAD)

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*aad*	Additional Authentication Data
in	*aad_len*	Length of AAD

10.46.2.5 atcac_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_decrypt_finish (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * tag,
            size_t tag_len,
            bool * is_verified )
```

Compare the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*tag*	GCM Tag to Verify
in	*tag_len*	Length of the GCM tag
out	*is_verified*	Tag verified as matching

10.46.2.6 atcac_aes_gcm_decrypt_start()

```
ATCA_STATUS atcac_aes_gcm_decrypt_start (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * key,
            const uint8_t key_len,
            const uint8_t * iv,
            const uint8_t iv_len )
```

Initialize an AES-GCM context for decryption.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.46.2.7 atcac_aes_gcm_decrypt_update()

```
ATCA_STATUS atcac_aes_gcm_decrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * ciphertext,
            const size_t ct_len,
            uint8_t * plaintext,
            size_t * pt_len )
```

Decrypt ciphertext using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*ciphertext*	Ciphertext to decrypt
in	*ct_len*	Length of the ciphertext
out	*plaintext*	Resulting decrypted plaintext
in,out	*pt_len*	Length of the plaintext buffer

10.46.2.8 atcac_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcac_aes_gcm_encrypt_finish (
            atcac_aes_gcm_ctx * ctx,
            uint8_t * tag,
            size_t tag_len )
```

Get the AES-GCM tag and free the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
out	*tag*	GCM Tag Result
in	*tag_len*	Length of the GCM tag

10.46.2.9 atcac_aes_gcm_encrypt_start()

```
ATCA_STATUS atcac_aes_gcm_encrypt_start (
            atcac_aes_gcm_ctx * ctx,
```

```
            const uint8_t * key,
            const uint8_t key_len,
            const uint8_t * iv,
            const uint8_t iv_len )
```

Initialize an AES-GCM context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*key*	AES Key
in	*key_len*	Length of the AES key - should be 16 or 32
in	*iv*	Initialization vector input
in	*iv_len*	Length of the initialization vector

10.46.2.10 atcac_aes_gcm_encrypt_update()

```
ATCA_STATUS atcac_aes_gcm_encrypt_update (
            atcac_aes_gcm_ctx * ctx,
            const uint8_t * plaintext,
            const size_t pt_len,
            uint8_t * ciphertext,
            size_t * ct_len )
```

Encrypt a data using the initialized context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	AES-GCM Context
in	*plaintext*	Input buffer to encrypt
in	*pt_len*	Length of the input
out	*ciphertext*	Output buffer
in,out	*ct_len*	Length of the ciphertext buffer

10.46.2.11 atcac_pk_derive()

```
ATCA_STATUS atcac_pk_derive (
            atcac_pk_ctx * private_ctx,
```

```
            atcac_pk_ctx * public_ctx,
            uint8_t * buf,
            size_t * buflen )
```

Execute the key agreement protocol for the provided keys (if they can)

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.46.2.12 atcac_pk_free()

```
ATCA_STATUS atcac_pk_free (
            atcac_pk_ctx * ctx )
```

Free a public/private key structure.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context

10.46.2.13 atcac_pk_init()

```
ATCA_STATUS atcac_pk_init (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t buflen,
            uint8_t key_type,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.46.2.14 atcac_pk_init_pem()

```
ATCA_STATUS atcac_pk_init_pem (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t buflen,
            bool pubkey )
```

Set up a public/private key structure for use in asymmetric cryptographic functions.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a pk context
in	*buf*	buffer containing a pem encoded key
in	*buflen*	length of the input buffer
in	*pubkey*	buffer is a public key

10.46.2.15 atcac_pk_public()

```
ATCA_STATUS atcac_pk_public (
            atcac_pk_ctx * ctx,
            uint8_t * buf,
            size_t * buflen )
```

Get the public key from the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.46.2.16 atcac_pk_sign()

```
ATCA_STATUS atcac_pk_sign (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t * sig_len )
```

Perform a signature with the private key in the context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.46.2.17 atcac_pk_verify()

```
ATCA_STATUS atcac_pk_verify (
            atcac_pk_ctx * ctx,
            uint8_t * digest,
            size_t dig_len,
            uint8_t * signature,
            size_t sig_len )
```

Perform a verify using the public key in the provided context.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.46.2.18 atcac_sw_sha1_finish()

```
int atcac_sw_sha1_finish (
            atcac_sha1_ctx * ctx,
            uint8_t digest[ATCA_SHA1_DIGEST_SIZE] )
```

Complete the SHA1 hash in software and return the digest.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
out	*digest*	output buffer (20 bytes)

10.46.2.19 atcac_sw_sha2_256_finish()

```
int atcac_sw_sha2_256_finish (
            atcac_sha2_256_ctx * ctx,
            uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE] )
```

Complete the SHA256 hash in software and return the digest.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*ctx*	pointer to a hash context
out	*digest*	output buffer (32 bytes)

10.47 atca_start_config.h File Reference

10.48 atca_start_iface.h File Reference

10.49 atca_status.h File Reference

Microchip Crypto Auth status codes.

```
#include <stdint.h>
#include "atca_bool.h"
```

Macros

- #define ATCA_STATUS_AUTH_BIT 0x40

Enumerations

- enum ATCA_STATUS {
  ATCA_SUCCESS = 0x00, ATCA_CONFIG_ZONE_LOCKED = 0x01, ATCA_DATA_ZONE_LOCKED = 0x02, ATCA_INVALID_POINTER,
  ATCA_INVALID_LENGTH, ATCA_WAKE_FAILED = 0xD0, ATCA_CHECKMAC_VERIFY_FAILED = 0xD1, ATCA_PARSE_ERROR = 0xD2,
  ATCA_STATUS_CRC = 0xD4, ATCA_STATUS_UNKNOWN = 0xD5, ATCA_STATUS_ECC = 0xD6, ATCA_STATUS_SELFTEST_ERROR = 0xD7,
  ATCA_FUNC_FAIL = 0xE0, ATCA_GEN_FAIL = 0xE1, ATCA_BAD_PARAM = 0xE2, ATCA_INVALID_ID = 0xE3,
  ATCA_INVALID_SIZE = 0xE4, ATCA_RX_CRC_ERROR = 0xE5, ATCA_RX_FAIL = 0xE6, ATCA_RX_NO_RESPONSE = 0xE7,
  ATCA_RESYNC_WITH_WAKEUP = 0xE8, ATCA_PARITY_ERROR = 0xE9, ATCA_TX_TIMEOUT = 0xEA, ATCA_RX_TIMEOUT = 0xEB,
  ATCA_TOO_MANY_COMM_RETRIES = 0xEC, ATCA_SMALL_BUFFER = 0xED, ATCA_COMM_FAIL = 0xF0, ATCA_TIMEOUT = 0xF1,
  ATCA_BAD_OPCODE = 0xF2, ATCA_WAKE_SUCCESS = 0xF3, ATCA_EXECUTION_ERROR = 0xF4, ATCA_UNIMPLEMENTED = 0xF5,
  ATCA_ASSERT_FAILURE = 0xF6, ATCA_TX_FAIL = 0xF7, ATCA_NOT_LOCKED = 0xF8, ATCA_NO_DEVICES = 0xF9,
  ATCA_HEALTH_TEST_ERROR = 0xFA, ATCA_ALLOC_FAILURE = 0xFB, ATCA_USE_FLAGS_CONSUMED = 0xFC, ATCA_NOT_INITIALIZED = 0xFD }

10.49.1 Detailed Description

Microchip Crypto Auth status codes.

Copyright

10.49.2 Macro Definition Documentation

10.49.2.1 ATCA_STATUS_AUTH_BIT

```
#define ATCA_STATUS_AUTH_BIT 0x40
```

10.49.3 Enumeration Type Documentation

10.49.3.1 ATCA_STATUS

```
enum ATCA_STATUS
```

Enumerator

ATCA_SUCCESS	Function succeeded.
ATCA_CONFIG_ZONE_LOCKED	
ATCA_DATA_ZONE_LOCKED	
ATCA_INVALID_POINTER	
ATCA_INVALID_LENGTH	
ATCA_WAKE_FAILED	response status byte indicates CheckMac failure (status byte = 0x01)
ATCA_CHECKMAC_VERIFY_FAILED	response status byte indicates CheckMac failure (status byte = 0x01)
ATCA_PARSE_ERROR	response status byte indicates parsing error (status byte = 0x03)
ATCA_STATUS_CRC	response status byte indicates DEVICE did not receive data properly (status byte = 0xFF)
ATCA_STATUS_UNKNOWN	response status byte is unknown
ATCA_STATUS_ECC	response status byte is ECC fault (status byte = 0x05)
ATCA_STATUS_SELFTEST_ERROR	response status byte is Self Test Error, chip in failure mode (status byte = 0x07)
ATCA_FUNC_FAIL	Function could not execute due to incorrect condition / state.
ATCA_GEN_FAIL	unspecified error
ATCA_BAD_PARAM	bad argument (out of range, null pointer, etc.)
ATCA_INVALID_ID	invalid device id, id not set
ATCA_INVALID_SIZE	Count value is out of range or greater than buffer size.

Enumerator

ATCA_RX_CRC_ERROR	CRC error in data received from device.
ATCA_RX_FAIL	Timed out while waiting for response. Number of bytes received is $>$ 0.
ATCA_RX_NO_RESPONSE	Not an error while the Command layer is polling for a command response.
ATCA_RESYNC_WITH_WAKEUP	Re-synchronization succeeded, but only after generating a Wake-up.
ATCA_PARITY_ERROR	for protocols needing parity
ATCA_TX_TIMEOUT	for Microchip PHY protocol, timeout on transmission waiting for master
ATCA_RX_TIMEOUT	for Microchip PHY protocol, timeout on receipt waiting for master
ATCA_TOO_MANY_COMM_RETRIES	Device did not respond too many times during a transmission. Could indicate no device present.
ATCA_SMALL_BUFFER	Supplied buffer is too small for data required.
ATCA_COMM_FAIL	Communication with device failed. Same as in hardware dependent modules.
ATCA_TIMEOUT	Timed out while waiting for response. Number of bytes received is 0.
ATCA_BAD_OPCODE	opcode is not supported by the device
ATCA_WAKE_SUCCESS	received proper wake token
ATCA_EXECUTION_ERROR	chip was in a state where it could not execute the command, response status byte indicates command execution error (status byte = 0x0F)
ATCA_UNIMPLEMENTED	Function or some element of it hasn't been implemented yet.
ATCA_ASSERT_FAILURE	Code failed run-time consistency check.
ATCA_TX_FAIL	Failed to write.
ATCA_NOT_LOCKED	required zone was not locked
ATCA_NO_DEVICES	For protocols that support device discovery (kit protocol), no devices were found.
ATCA_HEALTH_TEST_ERROR	random number generator health test error
ATCA_ALLOC_FAILURE	Couldn't allocate required memory.
ATCA_USE_FLAGS_CONSUMED	Use flags on the device indicates its consumed fully.
ATCA_NOT_INITIALIZED	The library has not been initialized so the command could not be executed.

10.50 atca_utils_sizes.c File Reference

API to Return structure sizes of cryptoauthlib structures.

```
#include "cryptoauthlib.h"
#include "atcacert/atcacert_date.h"
#include "atcacert/atcacert_def.h"
#include "host/atca_host.h"
```

Macros

- #define SIZE_OF_API_T(x) size_t x ## _size(void); size_t x ## _size(void) { return sizeof(x); }
- #define SIZE_OF_API_S(x) size_t x ## _size(void); size_t x ## _size(void) { return sizeof(struct x); }

Functions

- size_t atcacert_tm_utc_t_size (void)
- size_t atcacert_date_format_t_size (void)
- size_t atcacert_cert_type_t_size (void)
- size_t atcacert_cert_sn_src_t_size (void)
- size_t atcacert_device_zone_t_size (void)
- size_t atcacert_std_cert_element_t_size (void)
- size_t atcacert_device_loc_t_size (void)
- size_t atcacert_cert_loc_t_size (void)
- size_t atcacert_cert_element_t_size (void)
- size_t atcacert_def_t_size (void)
- size_t atcacert_build_state_t_size (void)
- size_t atca_aes_cbc_ctx_t_size (void)
- size_t atca_aes_cmac_ctx_t_size (void)
- size_t atca_aes_ctr_ctx_t_size (void)
- size_t atca_temp_key_t_size (void)
- size_t atca_include_data_in_out_size (void)
- size_t atca_nonce_in_out_t_size (void)
- size_t atca_io_decrypt_in_out_t_size (void)
- size_t atca_verify_mac_in_out_t_size (void)
- size_t atca_secureboot_enc_in_out_t_size (void)
- size_t atca_secureboot_mac_in_out_t_size (void)
- size_t atca_mac_in_out_t_size (void)
- size_t atca_hmac_in_out_size (void)
- size_t atca_gen_dig_in_out_t_size (void)
- size_t atca_write_mac_in_out_t_size (void)
- size_t atca_derive_key_in_out_size (void)
- size_t atca_derive_key_mac_in_out_size (void)
- size_t atca_decrypt_in_out_size (void)
- size_t atca_check_mac_in_out_t_size (void)
- size_t atca_verify_in_out_t_size (void)
- size_t atca_gen_key_in_out_t_size (void)
- size_t atca_sign_internal_in_out_t_size (void)
- size_t bool_size (void)
- size_t ATCAPacket_size (void)
- size_t atca_device_size (void)
- size_t ATCADeviceType_size (void)
- size_t ATCAIfaceType_size (void)
- size_t ATCAIfaceCfg_size (void)
- size_t atca_iface_size (void)
- size_t ATCA_STATUS_size (void)

10.50.1 Detailed Description

API to Return structure sizes of cryptoauthlib structures.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.50.2 Macro Definition Documentation

10.50.2.1 SIZE_OF_API_S

```
#define SIZE_OF_API_S(
             x ) size_t x ## _size(void); size_t x ## _size(void) { return sizeof(struct x );
}
```

10.50.2.2 SIZE_OF_API_T

```
#define SIZE_OF_API_T(
             x ) size_t x ## _size(void); size_t x ## _size(void) { return sizeof( x ); }
```

10.50.3 Function Documentation

10.50.3.1 atca_aes_cbc_ctx_t_size()

```
size_t atca_aes_cbc_ctx_t_size (
            void  )
```

10.50.3.2 atca_aes_cmac_ctx_t_size()

```
size_t atca_aes_cmac_ctx_t_size (
            void  )
```

10.50.3.3 atca_aes_ctr_ctx_t_size()

```
size_t atca_aes_ctr_ctx_t_size (
            void  )
```

10.50.3.4 atca_check_mac_in_out_t_size()

```
size_t atca_check_mac_in_out_t_size (
            void  )
```

10.50.3.5 atca_decrypt_in_out_size()

```
size_t atca_decrypt_in_out_size (
            void  )
```

10.50.3.6 atca_derive_key_in_out_size()

```
size_t atca_derive_key_in_out_size (
            void  )
```

10.50.3.7 atca_derive_key_mac_in_out_size()

```
size_t atca_derive_key_mac_in_out_size (
            void  )
```

10.50.3.8 atca_device_size()

```
size_t atca_device_size (
            void  )
```

10.50.3.9 atca_gen_dig_in_out_t_size()

```
size_t atca_gen_dig_in_out_t_size (
            void  )
```

10.50.3.10 atca_gen_key_in_out_t_size()

```
size_t atca_gen_key_in_out_t_size (
            void  )
```

10.50.3.11 atca_hmac_in_out_size()

```
size_t atca_hmac_in_out_size (
            void  )
```

10.50.3.12 atca_iface_size()

```
size_t atca_iface_size (
            void )
```

10.50.3.13 atca_include_data_in_out_size()

```
size_t atca_include_data_in_out_size (
            void )
```

10.50.3.14 atca_io_decrypt_in_out_t_size()

```
size_t atca_io_decrypt_in_out_t_size (
            void )
```

10.50.3.15 atca_mac_in_out_t_size()

```
size_t atca_mac_in_out_t_size (
            void )
```

10.50.3.16 atca_nonce_in_out_t_size()

```
size_t atca_nonce_in_out_t_size (
            void )
```

10.50.3.17 atca_secureboot_enc_in_out_t_size()

```
size_t atca_secureboot_enc_in_out_t_size (
            void )
```

10.50.3.18 atca_secureboot_mac_in_out_t_size()

```
size_t atca_secureboot_mac_in_out_t_size (
            void )
```

10.50.3.19 atca_sign_internal_in_out_t_size()

```
size_t atca_sign_internal_in_out_t_size (
            void )
```

10.50.3.20 ATCA_STATUS_size()

```
size_t ATCA_STATUS_size (
            void )
```

10.50.3.21 atca_temp_key_t_size()

```
size_t atca_temp_key_t_size (
            void )
```

10.50.3.22 atca_verify_in_out_t_size()

```
size_t atca_verify_in_out_t_size (
            void )
```

10.50.3.23 atca_verify_mac_in_out_t_size()

```
size_t atca_verify_mac_in_out_t_size (
            void )
```

10.50.3.24 atca_write_mac_in_out_t_size()

```
size_t atca_write_mac_in_out_t_size (
            void )
```

10.50.3.25 atcacert_build_state_t_size()

```
size_t atcacert_build_state_t_size (
            void )
```

10.50.3.26 atcacert_cert_element_t_size()

```
size_t atcacert_cert_element_t_size (
            void  )
```

10.50.3.27 atcacert_cert_loc_t_size()

```
size_t atcacert_cert_loc_t_size (
            void  )
```

10.50.3.28 atcacert_cert_sn_src_t_size()

```
size_t atcacert_cert_sn_src_t_size (
            void  )
```

10.50.3.29 atcacert_cert_type_t_size()

```
size_t atcacert_cert_type_t_size (
            void  )
```

10.50.3.30 atcacert_date_format_t_size()

```
size_t atcacert_date_format_t_size (
            void  )
```

10.50.3.31 atcacert_def_t_size()

```
size_t atcacert_def_t_size (
            void  )
```

10.50.3.32 atcacert_device_loc_t_size()

```
size_t atcacert_device_loc_t_size (
            void  )
```

10.50.3.33 atcacert_device_zone_t_size()

```
size_t atcacert_device_zone_t_size (
            void )
```

10.50.3.34 atcacert_std_cert_element_t_size()

```
size_t atcacert_std_cert_element_t_size (
            void )
```

10.50.3.35 atcacert_tm_utc_t_size()

```
size_t atcacert_tm_utc_t_size (
            void )
```

10.50.3.36 ATCADeviceType_size()

```
size_t ATCADeviceType_size (
            void )
```

10.50.3.37 ATCAIfaceCfg_size()

```
size_t ATCAIfaceCfg_size (
            void )
```

10.50.3.38 ATCAIfaceType_size()

```
size_t ATCAIfaceType_size (
            void )
```

10.50.3.39 ATCAPacket_size()

```
size_t ATCAPacket_size (
            void )
```

10.50.3.40 bool_size()

```
size_t bool_size (
            void )
```

10.51 atca_version.h File Reference

Microchip CryptoAuth Library Version.

Macros

- #define ATCA_LIBRARY_VERSION_DATE "20210126"
- #define ATCA_LIBRARY_VERSION_MAJOR 3
- #define ATCA_LIBRARY_VERSION_MINOR 3
- #define ATCA_LIBRARY_VERSION_BUILD 0

10.51.1 Detailed Description

Microchip CryptoAuth Library Version.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.51.2 Macro Definition Documentation

10.51.2.1 ATCA_LIBRARY_VERSION_BUILD

```
#define ATCA_LIBRARY_VERSION_BUILD 0
```

10.51.2.2 ATCA_LIBRARY_VERSION_DATE

```
#define ATCA_LIBRARY_VERSION_DATE "20210126"
```

10.51.2.3 ATCA_LIBRARY_VERSION_MAJOR

```
#define ATCA_LIBRARY_VERSION_MAJOR 3
```

10.51.2.4 ATCA_LIBRARY_VERSION_MINOR

```
#define ATCA_LIBRARY_VERSION_MINOR 3
```

10.52 atca_wolfssl_interface.c File Reference

Crypto abstraction functions for external host side cryptography.

```
#include "atca_config.h"
#include "atca_status.h"
#include "crypto/atca_crypto_sw.h"
```

10.52.1 Detailed Description

Crypto abstraction functions for external host side cryptography.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.53 atcacert.h File Reference

Declarations common to all atcacert code.

```
#include <stddef.h>
#include <stdint.h>
```

Macros

- #define FALSE (0)
- #define TRUE (1)
- #define ATCACERT_E_SUCCESS 0

  Operation completed successfully.
- #define ATCACERT_E_ERROR 1

  General error.
- #define ATCACERT_E_BAD_PARAMS 2

  Invalid/bad parameter passed to function.
- #define ATCACERT_E_BUFFER_TOO_SMALL 3

  Supplied buffer for output is too small to hold the result.
- #define ATCACERT_E_DECODING_ERROR 4

  Data being decoded/parsed has an invalid format.
- #define ATCACERT_E_INVALID_DATE 5

  Date is invalid.
- #define ATCACERT_E_UNIMPLEMENTED 6

  Function is unimplemented for the current configuration.

- #define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7

  A certificate element size was not what was expected.
- #define ATCACERT_E_ELEM_MISSING 8

  The certificate element isn't defined for the certificate definition.
- #define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9

  Certificate element is out of bounds for the given certificate.
- #define ATCACERT_E_BAD_CERT 10

  Certificate structure is bad in some way.
- #define ATCACERT_E_WRONG_CERT_DEF 11
- #define ATCACERT_E_VERIFY_FAILED 12

  Certificate or challenge/response verification failed.
- #define ATCACERT_E_INVALID_TRANSFORM 13

  Invalid transform passed to function.

10.53.1 Detailed Description

Declarations common to all atcacert code.

These are common definitions used by all the atcacert code.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.54 atcacert_client.c File Reference

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

```
#include <stdlib.h>
#include "atcacert_client.h"
#include "atcacert_der.h"
#include "atcacert_pem.h"
#include "cryptoauthlib.h"
#include "calib/calib_basic.h"
```

Functions

- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_↩ t response[64])

  Calculates the response to a challenge sent from the host.
- int atcacert_read_device_loc (const atcacert_device_loc_t ∗device_loc, uint8_t ∗data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t ∗cert_def, const uint8_t ca_public_key[64], uint8_t ∗cert, size_t ∗cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size)

Take a full certificate and write it to the ATECC508A device according to the certificate definition.

- int atcacert_create_csr_pem (const atcacert_def_t ∗csr_def, char ∗csr, size_t ∗csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_create_csr (const atcacert_def_t ∗csr_def, uint8_t ∗csr, size_t ∗csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_read_subj_key_id (const atcacert_def_t ∗cert_def, uint8_t subj_key_id[20])

  Reads the subject key ID based on a certificate definition.
- int atcacert_read_cert_size (const atcacert_def_t ∗cert_def, size_t ∗cert_size)

  Return the actual certificate size in bytes for a given cert def. Certificate can be variable size, so this gives the absolute buffer size when reading the certificates.

10.54.1 Detailed Description

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.55 atcacert_client.h File Reference

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_read_device_loc (const atcacert_device_loc_t ∗device_loc, uint8_t ∗data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t ∗cert_def, const uint8_t ca_public_key[64], uint8_t ∗cert, size_t ∗cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size)

  Take a full certificate and write it to the ATECC508A device according to the certificate definition.
- int atcacert_create_csr (const atcacert_def_t ∗csr_def, uint8_t ∗csr, size_t ∗csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_create_csr_pem (const atcacert_def_t ∗csr_def, char ∗csr, size_t ∗csr_size)

Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_t response[64])

  Calculates the response to a challenge sent from the host.
- int atcacert_read_subj_key_id (const atcacert_def_t *cert_def, uint8_t subj_key_id[20])

  Reads the subject key ID based on a certificate definition.
- int atcacert_read_cert_size (const atcacert_def_t *cert_def, size_t *cert_size)

  Return the actual certificate size in bytes for a given cert def. Certificate can be variable size, so this gives the absolute buffer size when reading the certificates.

10.55.1 Detailed Description

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.56 atcacert_date.c File Reference

Date handling with regard to certificates.

```
#include <string.h>
#include "atcacert_date.h"
```

Functions

- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t *timestamp, uint8_t *formatted_date, size_t *formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t *formatted_date, size_t formatted_date_size, atcacert_tm_utc_t *timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t *timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(20)])
- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[(20)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(13)])
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[(13)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(15)])
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[(15)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t *issue_date, uint8_t expire_years, uint8_t enc_dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t *issue_date, atcacert_tm_utc_t *expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [5]

10.56.1 Detailed Description

Date handling with regard to certificates.

Copyright

10.57 atcacert_date.h File Reference

Declarations for date handling with regard to certificates.

```
#include <stddef.h>
#include "atcacert.h"
```

Data Structures

- struct atcacert_tm_utc_s

Macros

- #define DATEFMT_ISO8601_SEP 0

  ISO8601 full date YYYY-MM-DDThh:mm:ssZ.
- #define DATEFMT_RFC5280_UTC 1

  RFC 5280 (X.509) 4.1.2.5.1 UTCTime format YYMMDDhhmmssZ.
- #define DATEFMT_POSIX_UINT32_BE 2

  POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, big endian.
- #define DATEFMT_POSIX_UINT32_LE 3

  POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, little endian.
- #define DATEFMT_RFC5280_GEN 4

  RFC 5280 (X.509) 4.1.2.5.2 GeneralizedTime format YYYYMMDDhhmmssZ.
- #define DATEFMT_ISO8601_SEP_SIZE (20)
- #define DATEFMT_RFC5280_UTC_SIZE (13)
- #define DATEFMT_POSIX_UINT32_BE_SIZE (4)
- #define DATEFMT_POSIX_UINT32_LE_SIZE (4)
- #define DATEFMT_RFC5280_GEN_SIZE (15)
- #define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
- #define ATCACERT_DATE_FORMAT_SIZES_COUNT 5

Typedefs

- typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
- typedef uint8_t atcacert_date_format_t

Functions

- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t *timestamp, uint8_t *formatted_date, size_t *formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t *formatted_date, size_t formatted_date_size, atcacert_tm_utc_t *timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t *issue_date, uint8_t expire_years, uint8_t enc_dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t *issue_date, atcacert_tm_utc_t *expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t *timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(20)])
- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[(20)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(13)])
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[(13)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(15)])
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[(15)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[(4)])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[(4)], atcacert_tm_utc_t *timestamp)

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [5]

10.57.1 Detailed Description

Declarations for date handling with regard to certificates.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.58 atcacert_def.c File Reference

Main certificate definition implementation.

```
#include "atcacert_def.h"
#include "crypto/atca_crypto_sw_sha1.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "atcacert_der.h"
#include "atcacert_date.h"
#include <string.h>
#include "atca_helpers.h"
```

Macros

- #define ATCACERT_MIN(x, y) ((x) < (y) ? (x) : (y))
- #define ATCACERT_MAX(x, y) ((x) >= (y) ? (x) : (y))

Functions

- int atcacert_merge_device_loc (atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_t device_locs_max_count, const atcacert_device_loc_t *device_loc, size_t block_size)

  Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.
- int atcacert_get_device_locs (const atcacert_def_t *cert_def, atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t *build_state, const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, const uint8_t ca_public_key[64])

  Starts the certificate rebuilding process.
- int atcacert_cert_build_process (atcacert_build_state_t *build_state, const atcacert_device_loc_t *device_loc, const uint8_t *device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t *build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.
- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t *device_loc1, const atcacert_device_loc_t *device_loc2)

  Determines if the two device locations overlap.
- int atcacert_get_device_data (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const atcacert_device_loc_t *device_loc, uint8_t *device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.
- int atcacert_set_subj_public_key (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t subj_public_key[64])

  Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t subj_public_key[64])

  Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t subj_key_id[20])

  Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, size_t max_cert_size, const uint8_t signature[64])

  Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t signature[64])

  Gets the signature from a certificate.
- int atcacert_set_issue_date (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const atcacert_tm_utc_t *timestamp)

  Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, atcacert_tm_utc_t *timestamp)

  Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.

- int atcacert_set_expire_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t signer_id[2])

  Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signer_id[2])

  Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.
- int atcacert_set_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t ∗cert_sn, size_t cert_sn_size)

  Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t device_sn[9])

  Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.
- int atcacert_get_cert_sn (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t ∗cert_sn, size_t ∗cert_sn_size)

  Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t auth_public_key[64])

  Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.
- int atcacert_set_auth_key_id_raw (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t ∗auth_key_id)

  Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t auth_key_id[20])

  Gets the authority key ID from a certificate.
- int atcacert_set_comp_cert (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t comp_cert[72])

  Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.
- int atcacert_get_comp_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t comp_cert[72])

  Generate the compressed certificate for the given certificate.
- int atcacert_get_tbs (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ∗∗tbs, size_t ∗tbs_size)

  Get a pointer to the TBS data in a certificate.
- int atcacert_get_tbs_digest (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t tbs_digest[32])

  Get the SHA256 digest of certificate's TBS data.
- int atcacert_set_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, uint8_t ∗cert, size_t cert_size, const uint8_t ∗data, size_t data_size)

  Sets an element in a certificate. The data_size must match the size in cert_loc.
- int atcacert_get_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, const uint8_t ∗cert, size_t cert_size, uint8_t ∗data, size_t data_size)

  Gets an element from a certificate.
- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

  Calculates the key ID for a given public ECC P256 key.

- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

  Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.
- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

  Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.
- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t ∗data, size_t data_size, uint8_t ∗destination, size_t ∗destination_size)

  Apply the specified transform to the specified data.
- int atcacert_max_cert_size (const atcacert_def_t ∗cert_def, size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

10.58.1 Detailed Description

Main certificate definition implementation.

Copyright

10.58.2 Macro Definition Documentation

10.58.2.1 ATCACERT_MAX

```
#define ATCACERT_MAX(
              x,
              y ) ((x) >= (y) ? (x) : (y))
```

10.58.2.2 ATCACERT_MIN

```
#define ATCACERT_MIN(
              x,
              y ) ((x) < (y) ? (x) : (y))
```

10.59 atcacert_def.h File Reference

Declarations for certificates related to ECC CryptoAuthentication devices. These are the definitions required to define a certificate and its various elements with regards to the CryptoAuthentication ECC devices.

```
#include <stddef.h>
#include <stdint.h>
#include "atca_compiler.h"
#include "atcacert.h"
#include "atcacert_date.h"
#include "atca_helpers.h"
```

Data Structures

- struct atcacert_device_loc_s
- struct atcacert_cert_loc_s
- struct atcacert_cert_element_s
- struct atcacert_def_s
- struct atcacert_build_state_s

Macros

- #define ATCA_MAX_TRANSFORMS 2
- #define ATCA_PACKED

Typedefs

- typedef enum atcacert_cert_type_e atcacert_cert_type_t
- typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
- typedef enum atcacert_device_zone_e atcacert_device_zone_t
- typedef enum atcacert_transform_e atcacert_transform_t

  How to transform the data from the device to the certificate.
- typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
- typedef struct atcacert_device_loc_s atcacert_device_loc_t
- typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
- typedef struct atcacert_cert_element_s atcacert_cert_element_t
- typedef struct atcacert_def_s atcacert_def_t
- typedef struct atcacert_build_state_s atcacert_build_state_t

Enumerations

- enum atcacert_cert_type_e { CERTTYPE_X509, CERTTYPE_CUSTOM }
- enum atcacert_cert_sn_src_e {
  SNSRC_STORED = 0x0, SNSRC_STORED_DYNAMIC = 0x7, SNSRC_DEVICE_SN = 0x8, SNSRC_SIGNER_ID = 0x9,
  SNSRC_PUB_KEY_HASH = 0xA, SNSRC_DEVICE_SN_HASH = 0xB, SNSRC_PUB_KEY_HASH_POS = 0xC, SNSRC_DEVICE_SN_HASH_POS = 0xD,
  SNSRC_PUB_KEY_HASH_RAW = 0xE, SNSRC_DEVICE_SN_HASH_RAW = 0xF }
- enum atcacert_device_zone_e { DEVZONE_CONFIG = 0x00, DEVZONE_OTP = 0x01, DEVZONE_DATA = 0x02, DEVZONE_NONE = 0x07 }
- enum atcacert_transform_e {
  TF_NONE, TF_REVERSE, TF_BIN2HEX_UC, TF_BIN2HEX_LC,
  TF_HEX2BIN_UC, TF_HEX2BIN_LC, TF_BIN2HEX_SPACE_UC, TF_BIN2HEX_SPACE_LC,
  TF_HEX2BIN_SPACE_UC, TF_HEX2BIN_SPACE_LC }

  How to transform the data from the device to the certificate.
- enum atcacert_std_cert_element_e {
  STDCERT_PUBLIC_KEY, STDCERT_SIGNATURE, STDCERT_ISSUE_DATE, STDCERT_EXPIRE_DATE,
  STDCERT_SIGNER_ID, STDCERT_CERT_SN, STDCERT_AUTH_KEY_ID, STDCERT_SUBJ_KEY_ID,
  STDCERT_NUM_ELEMENTS }

Functions

- int atcacert_get_device_locs (const atcacert_def_t ∗cert_def, atcacert_device_loc_t ∗device_locs, size_t ∗device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t ∗build_state, const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, const uint8_t ca_public_key[64])

  Starts the certificate rebuilding process.
- int atcacert_cert_build_process (atcacert_build_state_t ∗build_state, const atcacert_device_loc_t ∗device_loc, const uint8_t ∗device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t ∗build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.
- int atcacert_get_device_data (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const atcacert_device_loc_t ∗device_loc, uint8_t ∗device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.
- int atcacert_set_subj_public_key (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t subj_public_key[64])

  Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_public_key[64])

  Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_key_id[20])

  Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t signature[64])

  Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signature[64])

  Gets the signature from a certificate.
- int atcacert_set_issue_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_expire_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t signer_id[2])

  Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signer_id[2])

  Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.

- int atcacert_set_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t ∗cert_sn, size_t cert_sn_size)

  Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t device_sn[9])

  Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.
- int atcacert_get_cert_sn (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t ∗cert_sn, size_t ∗cert_sn_size)

  Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t auth_public_key[64])

  Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.
- int atcacert_set_auth_key_id_raw (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t ∗auth_key_id)

  Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t auth_key_id[20])

  Gets the authority key ID from a certificate.
- int atcacert_set_comp_cert (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t comp_cert[72])

  Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.
- int atcacert_get_comp_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t comp_cert[72])

  Generate the compressed certificate for the given certificate.
- int atcacert_get_tbs (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ∗∗tbs, size_t ∗tbs_size)

  Get a pointer to the TBS data in a certificate.
- int atcacert_get_tbs_digest (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t tbs_digest[32])

  Get the SHA256 digest of certificate's TBS data.
- int atcacert_set_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, uint8_t ∗cert, size_t cert_size, const uint8_t ∗data, size_t data_size)

  Sets an element in a certificate. The data_size must match the size in cert_loc.
- int atcacert_get_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, const uint8_t ∗cert, size_t cert_size, uint8_t ∗data, size_t data_size)

  Gets an element from a certificate.
- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

  Calculates the key ID for a given public ECC P256 key.
- int atcacert_merge_device_loc (atcacert_device_loc_t ∗device_locs, size_t ∗device_locs_count, size_t device_locs_max_count, const atcacert_device_loc_t ∗device_loc, size_t block_size)

  Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.
- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t ∗device_loc1, const atcacert_device_loc_t ∗device_loc2)

  Determines if the two device locations overlap.
- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

  Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.
- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

  Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.

- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t *data, size_t data_size, uint8_t *destination, size_t *destination_size)

  Apply the specified transform to the specified data.
- int atcacert_max_cert_size (const atcacert_def_t *cert_def, size_t *max_cert_size)

  Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

10.59.1 Detailed Description

Declarations for certificates related to ECC CryptoAuthentication devices. These are the definitions required to define a certificate and its various elements with regards to the CryptoAuthentication ECC devices.

Only the dynamic elements of a certificate (the parts of the certificate that change from device to device) are stored on the ATECC device. The definitions here describe the form of the certificate, and where the dynamic elements can be found both on the ATECC device itself and in the certificate template.

This also defines utility functions for working with the certificates and their definitions.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.59.2 Macro Definition Documentation

10.59.2.1 ATCA_MAX_TRANSFORMS

```
#define ATCA_MAX_TRANSFORMS 2
```

10.60 atcacert_der.c File Reference

functions required to work with DER encoded data related to X.509 certificates.

```
#include "atcacert_der.h"
#include <string.h>
```

Functions

- int atcacert_der_enc_length (uint32_t length, uint8_t *der_length, size_t *der_length_size)

  Encode a length in DER format.
- int atcacert_der_dec_length (const uint8_t *der_length, size_t *der_length_size, uint32_t *length)

  Decode a DER format length.
- int atcacert_der_adjust_length (uint8_t *der_length, size_t *der_length_size, int delta_length, uint32_t *new_length)
- int atcacert_der_enc_integer (const uint8_t *int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t *der_int, size_t *der_int_size)

  Encode an ASN.1 integer in DER format, including tag and length fields.
- int atcacert_der_dec_integer (const uint8_t *der_int, size_t *der_int_size, uint8_t *int_data, size_t *int_data_size)

  Decode an ASN.1 DER encoded integer.
- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t *der_sig, size_t *der_sig_size)

  Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t *der_sig, size_t *der_sig_size, uint8_t raw_sig[64])

  Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

10.60.1 Detailed Description

functions required to work with DER encoded data related to X.509 certificates.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.61 atcacert_der.h File Reference

function declarations required to work with DER encoded data related to X.509 certificates.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert.h"
```

Functions

- int atcacert_der_enc_length (uint32_t length, uint8_t *der_length, size_t *der_length_size)

  Encode a length in DER format.
- int atcacert_der_dec_length (const uint8_t *der_length, size_t *der_length_size, uint32_t *length)

  Decode a DER format length.
- int atcacert_der_adjust_length (uint8_t *der_length, size_t *der_length_size, int delta_length, uint32_t *new_length)
- int atcacert_der_enc_integer (const uint8_t *int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t *der_int, size_t *der_int_size)

  Encode an ASN.1 integer in DER format, including tag and length fields.
- int atcacert_der_dec_integer (const uint8_t *der_int, size_t *der_int_size, uint8_t *int_data, size_t *int_data_size)

  Decode an ASN.1 DER encoded integer.
- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t *der_sig, size_t *der_sig_size)

  Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t *der_sig, size_t *der_sig_size, uint8_t raw_sig[64])

  Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

10.61.1 Detailed Description

function declarations required to work with DER encoded data related to X.509 certificates.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.62 atcacert_host_hw.c File Reference

host side methods using CryptoAuth hardware

```
#include "atcacert_host_hw.h"
#include "atca_basic.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Functions

- int atcacert_verify_cert_hw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

    Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

    Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

    Verify a client's response to a challenge using the host's ATECC device for crypto functions.

10.62.1 Detailed Description

host side methods using CryptoAuth hardware

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.63 atcacert_host_hw.h File Reference

host side methods using CryptoAuth hardware

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_verify_cert_hw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

    Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

    Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

    Verify a client's response to a challenge using the host's ATECC device for crypto functions.

10.63.1 Detailed Description

host side methods using CryptoAuth hardware

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.64 atcacert_host_sw.c File Reference

host side methods using software implementations

```
#include "atcacert_host_sw.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "crypto/atca_crypto_sw_ecdsa.h"
#include "crypto/atca_crypto_sw_rand.h"
```

Functions

- int atcacert_verify_cert_sw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

10.64.1 Detailed Description

host side methods using software implementations

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.65 atcacert_host_sw.h File Reference

Host side methods using software implementations. host-side, the one authenticating a client, of the authentication process. Crypto functions are performed using a software library.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_verify_cert_sw (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

10.65.1 Detailed Description

Host side methods using software implementations. host-side, the one authenticating a client, of the authentication process. Crypto functions are performed using a software library.

Copyright

10.66 atcacert_pem.c File Reference

Functions required to work with PEM encoded data related to X.509 certificates.

```
#include <string.h>
#include "atcacert.h"
#include "atcacert_pem.h"
#include "atca_helpers.h"
```

Functions

- int atcacert_encode_pem (const uint8_t ∗der, size_t der_size, char ∗pem, size_t ∗pem_size, const char ∗header, const char ∗footer)

  Encode a DER data in PEM format.
- int atcacert_decode_pem (const char ∗pem, size_t pem_size, uint8_t ∗der, size_t ∗der_size, const char ∗header, const char ∗footer)

  Decode PEM data into DER format.
- int atcacert_encode_pem_cert (const uint8_t ∗der_cert, size_t der_cert_size, char ∗pem_cert, size_t ∗pem_cert_size)

  Encode a DER certificate in PEM format.
- int atcacert_encode_pem_csr (const uint8_t ∗der_csr, size_t der_csr_size, char ∗pem_csr, size_t ∗pem_csr_size)

  Encode a DER CSR in PEM format.
- int atcacert_decode_pem_cert (const char ∗pem_cert, size_t pem_cert_size, uint8_t ∗der_cert, size_t ∗der_cert_size)

  Decode a PEM certificate into DER format.
- int atcacert_decode_pem_csr (const char ∗pem_csr, size_t pem_csr_size, uint8_t ∗der_csr, size_t ∗der_csr_size)

  Extract the CSR certificate bytes from a PEM encoded CSR certificate.

10.66.1 Detailed Description

Functions required to work with PEM encoded data related to X.509 certificates.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.66.2 Function Documentation

10.66.2.1 atcacert_decode_pem()

```
int atcacert_decode_pem (
            const char * pem,
            size_t pem_size,
            uint8_t * der,
            size_t * der_size,
            const char * header,
            const char * footer )
```

Decode PEM data into DER format.

Parameters

in	*pem*	PEM data to decode to DER.
in	*pem_size*	PEM data size in bytes.
out	*der*	DER data is returned here.
in,out	*der_size*	As input, the size of the der buffer. As output, the size of the DER data.
in	*header*	Header to find the beginning of the PEM data.
in	*footer*	Footer to find the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.66.2.2 atcacert_decode_pem_cert()

```
int atcacert_decode_pem_cert (
            const char * pem_cert,
            size_t pem_cert_size,
            uint8_t * der_cert,
            size_t * der_cert_size )
```

Decode a PEM certificate into DER format.

Parameters

in	*pem_cert*	PEM certificate to decode to DER.
in	*pem_cert_size*	PEM certificate size in bytes.
out	*der_cert*	DER certificate is returned here.
in,out	*der_cert_size*	As input, the size of the der_cert buffer. As output, the size of the DER certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.66.2.3 atcacert_decode_pem_csr()

```
int atcacert_decode_pem_csr (
            const char * pem_csr,
            size_t pem_csr_size,
            uint8_t * der_csr,
            size_t * der_csr_size )
```

Extract the CSR certificate bytes from a PEM encoded CSR certificate.

Parameters

in	*pem_csr*	PEM CSR to decode to DER.
in	*pem_csr_size*	PEM CSR size in bytes.
out	*der_csr*	DER CSR is returned here.
in,out	*der_csr_size*	As input, the size of the der_csr buffer. As output, the size of the DER CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.66.2.4 atcacert_encode_pem()

```
int atcacert_encode_pem (
            const uint8_t * der,
            size_t der_size,
            char * pem,
            size_t * pem_size,
            const char * header,
            const char * footer )
```

Encode a DER data in PEM format.

Parameters

in	*der*	DER data to be encoded as PEM.
out	*der_size*	DER data size in bytes.
out	*pem*	PEM encoded data is returned here.
in,out	*pem_size*	As input, the size of the pem buffer. As output, the size of the PEM data.
in	*header*	Header to place at the beginning of the PEM data.
in	*footer*	Footer to place at the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.66.2.5 atcacert_encode_pem_cert()

```
int atcacert_encode_pem_cert (
            const uint8_t * der_cert,
            size_t der_cert_size,
            char * pem_cert,
            size_t * pem_cert_size )
```

Encode a DER certificate in PEM format.

Parameters

in	*der_cert*	DER certificate to be encoded as PEM.
out	*der_cert_size*	DER certificate size in bytes.
out	*pem_cert*	PEM encoded certificate is returned here.
in,out	*pem_cert_size*	As input, the size of the pem_cert buffer. As output, the size of the PEM certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.66.2.6 atcacert_encode_pem_csr()

```
int atcacert_encode_pem_csr (
            const uint8_t * der_csr,
            size_t der_csr_size,
            char * pem_csr,
            size_t * pem_csr_size )
```

Encode a DER CSR in PEM format.

Parameters

in	*der_csr*	DER CSR to be encoded as PEM.
out	*der_csr_size*	DER CSR size in bytes.
out	*pem_csr*	PEM encoded CSR is returned here.
in,out	*pem_csr_size*	As input, the size of the pem_csr buffer. As output, the size of the PEM CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67 atcacert_pem.h File Reference

Functions for converting between DER and PEM formats.

```
#include <stdint.h>
```

Macros

- #define PEM_CERT_BEGIN "-----BEGIN CERTIFICATE-----"
- #define PEM_CERT_END "-----END CERTIFICATE-----"
- #define PEM_CSR_BEGIN "-----BEGIN CERTIFICATE REQUEST-----"
- #define PEM_CSR_END "-----END CERTIFICATE REQUEST-----"

Functions

- int atcacert_encode_pem (const uint8_t ∗der, size_t der_size, char ∗pem, size_t ∗pem_size, const char ∗header, const char ∗footer)

    Encode a DER data in PEM format.
- int atcacert_decode_pem (const char ∗pem, size_t pem_size, uint8_t ∗der, size_t ∗der_size, const char ∗header, const char ∗footer)

    Decode PEM data into DER format.
- int atcacert_encode_pem_cert (const uint8_t ∗der_cert, size_t der_cert_size, char ∗pem_cert, size_t ∗pem↩_cert_size)

    Encode a DER certificate in PEM format.
- int atcacert_decode_pem_cert (const char ∗pem_cert, size_t pem_cert_size, uint8_t ∗der_cert, size_t ∗der↩_cert_size)

    Decode a PEM certificate into DER format.
- int atcacert_encode_pem_csr (const uint8_t ∗der_csr, size_t der_csr_size, char ∗pem_csr, size_t ∗pem_↩csr_size)

    Encode a DER CSR in PEM format.
- int atcacert_decode_pem_csr (const char ∗pem_csr, size_t pem_csr_size, uint8_t ∗der_csr, size_t ∗der_↩csr_size)

    Extract the CSR certificate bytes from a PEM encoded CSR certificate.

10.67.1 Detailed Description

Functions for converting between DER and PEM formats.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.67.2 Macro Definition Documentation

10.67.2.1 PEM_CERT_BEGIN

```
#define PEM_CERT_BEGIN "-----BEGIN CERTIFICATE-----"
```

10.67.2.2 PEM_CERT_END

```
#define PEM_CERT_END "-----END CERTIFICATE-----"
```

10.67.2.3 PEM_CSR_BEGIN

```
#define PEM_CSR_BEGIN "-----BEGIN CERTIFICATE REQUEST-----"
```

10.67.2.4 PEM_CSR_END

```
#define PEM_CSR_END "-----END CERTIFICATE REQUEST-----"
```

10.67.3 Function Documentation

10.67.3.1 atcacert_decode_pem()

```
int atcacert_decode_pem (
             const char * pem,
             size_t pem_size,
             uint8_t * der,
             size_t * der_size,
             const char * header,
             const char * footer )
```

Decode PEM data into DER format.

Parameters

in	*pem*	PEM data to decode to DER.
in	*pem_size*	PEM data size in bytes.
out	*der*	DER data is returned here.
in,out	*der_size*	As input, the size of the der buffer. As output, the size of the DER data.
in	*header*	Header to find the beginning of the PEM data.
in	*footer*	Footer to find the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67.3.2 atcacert_decode_pem_cert()

```
int atcacert_decode_pem_cert (
            const char * pem_cert,
            size_t pem_cert_size,
            uint8_t * der_cert,
            size_t * der_cert_size )
```

Decode a PEM certificate into DER format.

Parameters

in	*pem_cert*	PEM certificate to decode to DER.
in	*pem_cert_size*	PEM certificate size in bytes.
out	*der_cert*	DER certificate is returned here.
in,out	*der_cert_size*	As input, the size of the der_cert buffer. As output, the size of the DER certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67.3.3 atcacert_decode_pem_csr()

```
int atcacert_decode_pem_csr (
            const char * pem_csr,
            size_t pem_csr_size,
            uint8_t * der_csr,
            size_t * der_csr_size )
```

Extract the CSR certificate bytes from a PEM encoded CSR certificate.

Parameters

in	*pem_csr*	PEM CSR to decode to DER.
in	*pem_csr_size*	PEM CSR size in bytes.
out	*der_csr*	DER CSR is returned here.
in,out	*der_csr_size*	As input, the size of the der_csr buffer. As output, the size of the DER CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67.3.4 atcacert_encode_pem()

```
int atcacert_encode_pem (
            const uint8_t * der,
            size_t der_size,
            char * pem,
            size_t * pem_size,
            const char * header,
            const char * footer )
```

Encode a DER data in PEM format.

Parameters

in	*der*	DER data to be encoded as PEM.
out	*der_size*	DER data size in bytes.
out	*pem*	PEM encoded data is returned here.
in,out	*pem_size*	As input, the size of the pem buffer. As output, the size of the PEM data.
in	*header*	Header to place at the beginning of the PEM data.
in	*footer*	Footer to place at the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67.3.5 atcacert_encode_pem_cert()

```
int atcacert_encode_pem_cert (
            const uint8_t * der_cert,
            size_t der_cert_size,
            char * pem_cert,
            size_t * pem_cert_size )
```

Encode a DER certificate in PEM format.

Parameters

in	*der_cert*	DER certificate to be encoded as PEM.
out	*der_cert_size*	DER certificate size in bytes.
out	*pem_cert*	PEM encoded certificate is returned here.
in,out	*pem_cert_size*	As input, the size of the pem_cert buffer. As output, the size of the PEM certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.67.3.6 atcacert_encode_pem_csr()

```
int atcacert_encode_pem_csr (
            const uint8_t * der_csr,
            size_t der_csr_size,
            char * pem_csr,
            size_t * pem_csr_size )
```

Encode a DER CSR in PEM format.

Parameters

in	*der_csr*	DER CSR to be encoded as PEM.
out	*der_csr_size*	DER CSR size in bytes.
out	*pem_csr*	PEM encoded CSR is returned here.
in,out	*pem_csr_size*	As input, the size of the pem_csr buffer. As output, the size of the PEM CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.68 calib_aes.c File Reference

CryptoAuthLib Basic API methods for AES command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_aes (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t *aes_in, uint8_t *aes_out)

  Compute the AES-128 encrypt, decrypt, or GFM calculation.

- ATCA_STATUS calib_aes_encrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *plaintext, uint8_t *ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS calib_aes_decrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t *ciphertext, uint8_t *plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS calib_aes_gfm (ATCADevice device, const uint8_t *h, const uint8_t *input, uint8_t *output)

  Perform a Galois Field Multiply (GFM) operation.

10.68.1 Detailed Description

CryptoAuthLib Basic API methods for AES command.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A/B. Refer to device edatasheet for full details.

Copyright

10.69 calib_aes_gcm.c File Reference

CryptoAuthLib Basic API methods for AES GCM mode.

```
#include "cryptoauthlib.h"
#include "calib_aes_gcm.h"
```

- #define RETURN return ATCA_TRACE
- const char * atca_basic_aes_gcm_version = "2.0"
- ATCA_STATUS calib_aes_gcm_init (ATCADevice device, atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, const uint8_t *iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS calib_aes_gcm_init_rand (ATCADevice device, atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t *free_field, size_t free_field_size, uint8_t *iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS calib_aes_gcm_aad_update (ATCADevice device, atca_aes_gcm_ctx_t *ctx, const uint8_t *aad, uint32_t aad_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.
- ATCA_STATUS calib_aes_gcm_encrypt_update (ATCADevice device, atca_aes_gcm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS calib_aes_gcm_encrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t *ctx, uint8_t *tag, size_t tag_size)

Complete a GCM encrypt operation returning the authentication tag.

- ATCA_STATUS calib_aes_gcm_decrypt_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS calib_aes_gcm_decrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗tag, size_t tag_size, bool ∗is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.

10.69.1 Detailed Description

CryptoAuthLib Basic API methods for AES GCM mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A/B. Refer to device datasheet for full details.

Copyright

10.69.2 Macro Definition Documentation

10.69.2.1 RETURN

```
#define RETURN return ATCA_TRACE
```

10.69.3 Function Documentation

10.69.3.1 calib_aes_gcm_aad_update()

```
ATCA_STATUS calib_aes_gcm_aad_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * aad,
            uint32_t aad_size )
```

Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.

This can be called multiple times. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before atcab_aes_gcm_encrypt_update() or atcab_aes_gcm_decrypt_update().

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.2 calib_aes_gcm_decrypt_finish()

```
ATCA_STATUS calib_aes_gcm_decrypt_finish (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * tag,
            size_t tag_size,
            bool * is_verified )
```

Complete a GCM decrypt operation verifying the authentication tag.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*tag*	Expected authentication tag.
in	*tag_size*	Size of tag in bytes (12 to 16 bytes).
out	*is_verified*	Returns whether or not the tag verified.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.3 calib_aes_gcm_decrypt_update()

```
ATCA_STATUS calib_aes_gcm_decrypt_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*ciphertext*	Ciphertext to be decrypted.
in	*ciphertext_size*	Size of ciphertext in bytes.
out	*plaintext*	Decrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.4 calib_aes_gcm_encrypt_finish()

```
ATCA_STATUS calib_aes_gcm_encrypt_finish (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint8_t * tag,
            size_t tag_size )
```

Complete a GCM encrypt operation returning the authentication tag.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
out	*tag*	Authentication tag is returned here.
in	*tag_size*	Tag size in bytes (12 to 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.5 calib_aes_gcm_encrypt_update()

```
ATCA_STATUS calib_aes_gcm_encrypt_update (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
in	*plaintext_size*	Size of plaintext in bytes.
out	*ciphertext*	Encrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.6 calib_aes_gcm_init()

```
ATCA_STATUS calib_aes_gcm_init (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv,
            size_t iv_size )
```

Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES GCM context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector.
in	*iv_size*	Size of IV in bytes. Standard is 12 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.69.3.7 calib_aes_gcm_init_rand()

```
ATCA_STATUS calib_aes_gcm_init_rand (
            ATCADevice device,
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            size_t rand_size,
```

```
const uint8_t * free_field,
size_t free_field_size,
uint8_t * iv )
```

Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.

Parameters

in	*device*	Device context pointer
in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*rand_size*	Size of the random field in bytes. Minimum and recommended size is 12 bytes. Max is 32 bytes.
in	*free_field*	Fixed data to include in the IV after the random field. Can be NULL if not used.
in	*free_field_size*	Size of the free field in bytes.
out	*iv*	Initialization vector is returned here. Its size will be rand_size and free_field_size combined.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.70 calib_aes_gcm.h File Reference

Unity tests for the cryptoauthlib AES GCM functions.

Data Structures

- struct atca_aes_gcm_ctx

- #define ATCA_AES_GCM_IV_STD_LENGTH 12
- typedef struct atca_aes_gcm_ctx atca_aes_gcm_ctx_t
- const char ∗ atca_basic_aes_gcm_version
- ATCA_STATUS calib_aes_gcm_init (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8←_t key_block, const uint8_t ∗iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS calib_aes_gcm_init_rand (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t ∗free_field, size_t free_field_size, uint8_t ∗iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS calib_aes_gcm_aad_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗aad, uint32_t aad_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608 device.
- ATCA_STATUS calib_aes_gcm_encrypt_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

- ATCA_STATUS calib_aes_gcm_encrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, uint8_t ∗tag, size_t tag_size)

  Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS calib_aes_gcm_decrypt_update (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  Decrypt data using GCM mode and a key within the ATECC608 device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS calib_aes_gcm_decrypt_finish (ATCADevice device, atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗tag, size_t tag_size, bool ∗is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.

10.70.1 Detailed Description

Unity tests for the cryptoauthlib AES GCM functions.

Copyright

10.71 calib_basic.c File Reference

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_wakeup_i2c (ATCADevice device)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS calib_wakeup (ATCADevice device)

  wakeup the CryptoAuth device
- ATCA_STATUS calib_idle (ATCADevice device)

  idle the CryptoAuth device
- ATCA_STATUS calib_sleep (ATCADevice device)

  invoke sleep on the CryptoAuth device
- ATCA_STATUS _calib_exit (ATCADevice device)

  common cleanup code which idles the device after any operation
- ATCA_STATUS calib_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS calib_ecc204_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset for ECC204 device.
- ATCA_STATUS calib_get_zone_size (ATCADevice device, uint8_t zone, uint16_t slot, size_t ∗size)

  Gets the size of the specified zone in bytes.

10.71.1 Detailed Description

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

Copyright

10.71.2 Function Documentation

10.71.2.1 calib_wakeup_i2c()

```
ATCA_STATUS calib_wakeup_i2c (
            ATCADevice device )
```

basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.

10.72 calib_basic.h File Reference

```
#include "calib_command.h"
#include "calib_execution.h"
```

Data Structures

- struct atca_sha256_ctx

Typedefs

- typedef struct atca_sha256_ctx atca_sha256_ctx_t
- typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t

Functions

- ATCA_STATUS calib_wakeup (ATCADevice device)

    wakeup the CryptoAuth device
- ATCA_STATUS calib_idle (ATCADevice device)

    idle the CryptoAuth device
- ATCA_STATUS calib_sleep (ATCADevice device)

    invoke sleep on the CryptoAuth device
- ATCA_STATUS _calib_exit (ATCADevice device)

    common cleanup code which idles the device after any operation
- ATCA_STATUS calib_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

    Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS calib_get_zone_size (ATCADevice device, uint8_t zone, uint16_t slot, size_t ∗size)

    Gets the size of the specified zone in bytes.
- ATCA_STATUS calib_ecc204_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

    Compute the address given the zone, slot, block, and offset for ECC204 device.
- ATCA_STATUS calib_is_locked (ATCADevice device, uint8_t zone, bool ∗is_locked)

    Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS calib_is_slot_locked (ATCADevice device, uint16_t slot, bool ∗is_locked)

    Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS calib_is_private (ATCADevice device, uint16_t slot, bool ∗is_private)

    Check if a slot is a private key.
- ATCA_STATUS calib_ecc204_is_locked (ATCADevice device, uint8_t zone, bool ∗is_locked)
- ATCA_STATUS calib_ecc204_is_data_locked (ATCADevice device, bool ∗is_locked)
- ATCA_STATUS calib_ecc204_is_config_locked (ATCADevice device, bool ∗is_locked)
- ATCA_STATUS calib_aes (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗aes_in, uint8↩_t ∗aes_out)

    Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS calib_aes_encrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

    Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS calib_aes_decrypt (ATCADevice device, uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

    Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS calib_aes_gfm (ATCADevice device, const uint8_t ∗h, const uint8_t ∗input, uint8_t ∗output)

    Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS calib_checkmac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_↩t ∗challenge, const uint8_t ∗response, const uint8_t ∗other_data)

    Compares a MAC response with input values.
- ATCA_STATUS calib_counter (ATCADevice device, uint8_t mode, uint16_t counter_id, uint32_t ∗counter↩_value)

    Compute the Counter functions.
- ATCA_STATUS calib_counter_increment (ATCADevice device, uint16_t counter_id, uint32_t ∗counter_↩value)

    Increments one of the device's monotonic counters.
- ATCA_STATUS calib_counter_read (ATCADevice device, uint16_t counter_id, uint32_t ∗counter_value)

    Read one of the device's monotonic counters.
- ATCA_STATUS calib_derivekey (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗mac)

    Executes the DeviceKey command for deriving a new key from a nonce (TempKey) and an existing key.
- ATCA_STATUS calib_ecdh_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_↩t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)

Base function for generating premaster secret key using ECDH.

- ATCA_STATUS calib_ecdh (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_enc (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_ecdh_ioenc (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS calib_ecdh_tempkey (ATCADevice device, const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_tempkey_ioenc (ATCADevice device, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS calib_gendig (ATCADevice device, uint8_t zone, uint16_t key_id, const uint8_t ∗other_data, uint8_t other_data_size)

  Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS calib_genkey_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗other_data, uint8_t ∗public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS calib_genkey (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS calib_get_pubkey (ATCADevice device, uint16_t key_id, uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS calib_genkey_mac (ATCADevice device, uint8_t ∗public_key, uint8_t ∗mac)

  Uses Genkey command to calculate SHA256 digest MAC of combining public key and session key.
- ATCA_STATUS calib_hmac (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t ∗digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS calib_info_base (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS calib_info (ATCADevice device, uint8_t ∗revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS calib_info_set_latch (ATCADevice device, bool state)

  Use the Info command to set the persistent latch state for an ATECC608 device.
- ATCA_STATUS calib_info_get_latch (ATCADevice device, bool ∗state)

  Use the Info command to get the persistent latch current state for an ATECC608 device.
- ATCA_STATUS calib_info_privkey_valid (ATCADevice device, uint16_t key_id, uint8_t ∗is_valid)

  Use Info command to check ECC Private key stored in key slot is valid or not.
- ATCA_STATUS calib_info_lock_status (ATCADevice device, uint16_t param2, uint8_t ∗is_locked)
- ATCA_STATUS calib_kdf (ATCADevice device, uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

  Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS calib_lock (ATCADevice device, uint8_t mode, uint16_t summary_crc)

  The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS calib_lock_config_zone (ATCADevice device)

  Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS calib_lock_config_zone_crc (ATCADevice device, uint16_t summary_crc)

Lock the config zone with summary CRC.
- ATCA_STATUS calib_lock_data_zone (ATCADevice device)

  Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS calib_lock_data_zone_crc (ATCADevice device, uint16_t summary_crc)

  Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS calib_lock_data_slot (ATCADevice device, uint16_t slot)

  Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).
- ATCA_STATUS calib_ecc204_lock_config_slot (ATCADevice device, uint8_t slot, uint16_t summary_crc)

  Use Lock command to lock individual configuration zone slots.
- ATCA_STATUS calib_ecc204_lock_config_zone (ATCADevice device)

  Use lock command to lock complete configuration zone.
- ATCA_STATUS calib_ecc204_lock_data_slot (ATCADevice device, uint8_t slot)

  Use lock command to lock data zone slot.
- ATCA_STATUS calib_mac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS calib_nonce_base (ATCADevice device, uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS calib_nonce (ATCADevice device, const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_nonce_load (ATCADevice device, uint8_t target, const uint8_t ∗num_in, uint16_↩ t num_in_size)

  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS calib_nonce_rand (ATCADevice device, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_challenge (ATCADevice device, const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_challenge_seed_update (ATCADevice device, const uint8_t ∗num_in, uint8_t ∗rand↩ _out)

  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_nonce_gen_session_key (ATCADevice device, uint16_t param2, uint8_t ∗num_in, uint8_t ∗rand_out)

  Use Nonce command to generate session key for use by a subsequent write command This Mode only supports in ECC204 device.
- ATCA_STATUS calib_priv_write (ATCADevice device, uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[(20)])
- ATCA_STATUS calib_random (ATCADevice device, uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the CryptoAuth device.
- ATCA_STATUS calib_read_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_↩ t offset, uint8_t ∗data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS calib_read_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.
- ATCA_STATUS calib_read_serial_number (ATCADevice device, uint8_t ∗serial_number)

  Executes Read command, which reads the 9 byte serial number of the device from the config zone.

- ATCA_STATUS calib_read_pubkey (ATCADevice device, uint16_t slot, uint8_t ∗public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS calib_read_sig (ATCADevice device, uint16_t slot, uint8_t ∗sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS calib_read_config_zone (ATCADevice device, uint8_t ∗config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS calib_cmp_config_zone (ATCADevice device, uint8_t ∗config_data, bool ∗same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS calib_ecc204_read_zone (ATCADevice device, uint8_t zone, uint8_t slot, uint8_t block, size_t offset, uint8_t ∗data, uint8_t len)
- ATCA_STATUS calib_ecc204_read_config_zone (ATCADevice device, uint8_t ∗config_data)
- ATCA_STATUS calib_ecc204_read_serial_number (ATCADevice device, uint8_t ∗serial_number)
- ATCA_STATUS calib_ecc204_read_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_←
t block, uint8_t ∗data, size_t length)
- ATCA_STATUS calib_ecc204_cmp_config_zone (ATCADevice device, uint8_t ∗config_data, bool ∗same_←
config)
- ATCA_STATUS calib_read_enc (ATCADevice device, uint16_t key_id, uint8_t block, uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_secureboot (ATCADevice device, uint8_t mode, uint16_t param2, const uint8_←
t ∗digest, const uint8_t ∗signature, uint8_t ∗mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS calib_secureboot_mac (ATCADevice device, uint8_t mode, const uint8_t ∗digest, const uint8_t ∗signature, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS calib_selftest (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t ∗result)

  *Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the AT←
ECC608 chip.*
- ATCA_STATUS calib_sha_base (ATCADevice device, uint8_t mode, uint16_t length, const uint8_t ∗data_in, uint8_t ∗data_out, uint16_t ∗data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS calib_sha_start (ATCADevice device)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS calib_sha_update (ATCADevice device, const uint8_t ∗message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS calib_sha_end (ATCADevice device, uint8_t ∗digest, uint16_t length, const uint8_←
t ∗message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_read_context (ATCADevice device, uint8_t ∗context, uint16_t ∗context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS calib_sha_write_context (ATCADevice device, const uint8_t ∗context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.
- ATCA_STATUS calib_sha (ATCADevice device, uint16_t length, const uint8_t ∗message, uint8_t ∗digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_hw_sha2_256 (ATCADevice device, const uint8_t ∗data, size_t data_size, uint8_←
t ∗digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_hw_sha2_256_init (ATCADevice device, atca_sha256_ctx_t ∗ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS calib_hw_sha2_256_update (ATCADevice device, atca_sha256_ctx_t ∗ctx, const uint8_←
t ∗data, size_t data_size)

Add message data to a SHA context for performing a hardware SHA-256 operation on a device.

- ATCA_STATUS calib_hw_sha2_256_finish (ATCADevice device, atca_sha256_ctx_t *ctx, uint8_t *digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS calib_sha_hmac_init (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, uint16_t key_slot)

  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_update (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_finish (ATCADevice device, atca_hmac_sha256_ctx_t *ctx, uint8_t *digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac (ATCADevice device, const uint8_t *data, size_t data_size, uint16_t key_slot, uint8_t *digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS calib_sign_base (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t *signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS calib_sign (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_sign_internal (ATCADevice device, uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t *signature)

  Executes Sign command to sign an internally generated message.
- ATCA_STATUS calib_ecc204_sign (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  Execute sign command to sign the 32 bytes message digest using private key mentioned in slot.
- ATCA_STATUS calib_updateextra (ATCADevice device, uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).
- ATCA_STATUS calib_verify (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t *signature, const uint8_t *public_key, const uint8_t *other_data, uint8_t *mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS calib_verify_extern (ATCADevice device, const uint8_t *message, const uint8_t *signature, const uint8_t *public_key, bool *is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_verify_extern_mac (ATCADevice device, const uint8_t *message, const uint8_t *signature, const uint8_t *public_key, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_stored (ATCADevice device, const uint8_t *message, const uint8_t *signature, uint16_t key_id, bool *is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_verify_stored_mac (ATCADevice device, const uint8_t *message, const uint8_t *signature, uint16_t key_id, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_validate (ATCADevice device, uint16_t key_id, const uint8_t *signature, const uint8_t *other_data, bool *is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.

- ATCA_STATUS calib_verify_invalidate (ATCADevice device, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS calib_write (ATCADevice device, uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS calib_write_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS calib_write_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS calib_write_pubkey (ATCADevice device, uint16_t slot, const uint8_t ∗public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS calib_write_config_zone (ATCADevice device, const uint8_t ∗config_data)

  Executes the Write command, which writes the configuration zone.
- ATCA_STATUS calib_ecc204_write (ATCADevice device, uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)
- ATCA_STATUS calib_ecc204_write_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)
- ATCA_STATUS calib_ecc204_write_config_zone (ATCADevice device, const uint8_t ∗config_data)
- ATCA_STATUS calib_ecc204_write_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t block, const uint8_t ∗data, size_t length)
- ATCA_STATUS calib_write_enc (ATCADevice device, uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id, const uint8_t num_in[(20)])
- ATCA_STATUS calib_ecc204_write_enc (ATCADevice device, uint8_t slot, uint8_t ∗data, uint8_t ∗transport_key, uint8_t key_id, uint8_t num_in[(20)])
- ATCA_STATUS calib_write_config_counter (ATCADevice device, uint16_t counter_id, uint32_t counter_value)

  Initialize one of the monotonic counters in device with a specific value.

10.73 calib_checkmac.c File Reference

CryptoAuthLib Basic API methods for CheckMAC command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_checkmac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, const uint8_t ∗response, const uint8_t ∗other_data)

  Compares a MAC response with input values.

10.73.1 Detailed Description

CryptoAuthLib Basic API methods for CheckMAC command.

The CheckMac command calculates a MAC response that would have been generated on a different Crypto↩ Authentication device and then compares the result with input value.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.74 calib_command.c File Reference

Microchip CryptoAuthentication device command builder - this is the main object that builds the command byte strings for the given device. It does not execute the command. The basic flow is to call a command method to build the command you want given the parameters and then send that byte string through the device interface.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS atCheckMAC (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand CheckMAC method.
- ATCA_STATUS atCounter (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Counter method.
- ATCA_STATUS atDeriveKey (ATCADeviceType device_type, ATCAPacket ∗packet, bool has_mac)

  ATCACommand DeriveKey method.
- ATCA_STATUS atECDH (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand ECDH method.
- ATCA_STATUS atGenDig (ATCADeviceType device_type, ATCAPacket ∗packet, bool is_no_mac_key)

  ATCACommand Generate Digest method.
- ATCA_STATUS atGenKey (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Generate Key method.
- ATCA_STATUS atHMAC (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand HMAC method.
- ATCA_STATUS atInfo (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Info method.
- ATCA_STATUS atLock (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Lock method.
- ATCA_STATUS atMAC (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand MAC method.
- ATCA_STATUS atNonce (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Nonce method.
- ATCA_STATUS atPause (ATCADeviceType device_type, ATCAPacket ∗packet)

ATCACommand Pause method.
- ATCA_STATUS atPrivWrite (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand PrivWrite method.
- ATCA_STATUS atRandom (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Random method.
- ATCA_STATUS atRead (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Read method.
- ATCA_STATUS atSecureBoot (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand SecureBoot method.
- ATCA_STATUS atSHA (ATCADeviceType device_type, ATCAPacket ∗packet, uint16_t write_context_size)

  ATCACommand SHA method.
- ATCA_STATUS atSign (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Sign method.
- ATCA_STATUS atUpdateExtra (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand UpdateExtra method.
- ATCA_STATUS atVerify (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand ECDSA Verify method.
- ATCA_STATUS atWrite (ATCADeviceType device_type, ATCAPacket ∗packet, bool has_mac)

  ATCACommand Write method.
- ATCA_STATUS atAES (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atSelfTest (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atKDF (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand KDF method.
- void atCRC (size_t length, const uint8_t ∗data, uint8_t ∗crc_le)

  Calculates CRC over the given raw data and returns the CRC in little-endian byte order.
- void atCalcCrc (ATCAPacket ∗packet)

  This function calculates CRC and adds it to the correct offset in the packet data.
- ATCA_STATUS atCheckCrc (const uint8_t ∗response)

  This function checks the consistency of a response.
- bool atIsSHAFamily (ATCADeviceType device_type)

  determines if a given device type is a SHA device or a superset of a SHA device
- bool atIsECCFamily (ATCADeviceType device_type)

  determines if a given device type is an ECC device or a superset of a ECC device
- ATCA_STATUS isATCAError (uint8_t ∗data)

  checks for basic error frame in data

10.74.1 Detailed Description

Microchip CryptoAuthentication device command builder - this is the main object that builds the command byte strings for the given device. It does not execute the command. The basic flow is to call a command method to build the command you want given the parameters and then send that byte string through the device interface.

The primary goal of the command builder is to wrap the given parameters with the correct packet size and CRC. The caller should first fill in the parameters required in the ATCAPacket parameter given to the command. The command builder will deal with the mechanics of creating a valid packet using the parameter information.

Copyright

10.74.2 Function Documentation

10.74.2.1 atAES()

```
ATCA_STATUS atAES (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.2 atCalcCrc()

```
void atCalcCrc (
            ATCAPacket * packet )
```

This function calculates CRC and adds it to the correct offset in the packet data.

Parameters

in	*packet*	Packet to calculate CRC data for

10.74.2.3 atCheckCrc()

```
ATCA_STATUS atCheckCrc (
            const uint8_t * response )
```

This function checks the consistency of a response.

Parameters

in	*response*	pointer to response

Returns

ATCA_SUCCESS on success, otherwise ATCA_RX_CRC_ERROR

10.74.2.4 atCheckMAC()

```
ATCA_STATUS atCheckMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand CheckMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.5 atCounter()

```
ATCA_STATUS atCounter (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Counter method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.74.2.6 atCRC()

```
void atCRC (
            size_t length,
```

```
const uint8_t * data,
uint8_t * crc_le )
```

Calculates CRC over the given raw data and returns the CRC in little-endian byte order.

Parameters

in	*length*	Size of data not including the CRC byte positions
in	*data*	Pointer to the data over which to compute the CRC
out	*crc_le*	Pointer to the place where the two-bytes of CRC will be returned in little-endian byte order.

10.74.2.7 atDeriveKey()

```
ATCA_STATUS atDeriveKey (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand DeriveKey method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	hasMAC determines if MAC data is present in the packet input

Returns

ATCA_SUCCESS

10.74.2.8 atECDH()

```
ATCA_STATUS atECDH (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand ECDH method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.9 atGenDig()

```
ATCA_STATUS atGenDig (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool is_no_mac_key )
```

ATCACommand Generate Digest method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*is_no_mac_key*	Should be true if GenDig is being run on a slot that has its SlotConfig.NoMac bit set

Returns

ATCA_SUCCESS

10.74.2.10 atGenKey()

```
ATCA_STATUS atGenKey (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Generate Key method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.11 atHMAC()

```
ATCA_STATUS atHMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand HMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.12 atInfo()

```
ATCA_STATUS atInfo (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Info method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.13 atIsECCFamily()

```
bool atIsECCFamily (
            ATCADeviceType device_type )
```

determines if a given device type is an ECC device or a superset of a ECC device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is an ECC family device.

10.74.2.14 atIsSHAFamily()

```
bool atIsSHAFamily (
            ATCADeviceType device_type )
```

determines if a given device type is a SHA device or a superset of a SHA device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is a SHA family device.

10.74.2.15 atKDF()

```
ATCA_STATUS atKDF (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand KDF method.

Parameters

in	*ca_cmd*	Instance
in	*packet*	Pointer to the packet containing the command being built.

Returns

ATCA_SUCCESS

10.74.2.16 atLock()

```
ATCA_STATUS atLock (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Lock method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.17 atMAC()

```
ATCA_STATUS atMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand MAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.18 atNonce()

```
ATCA_STATUS atNonce (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Nonce method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.74.2.19 atPause()

```
ATCA_STATUS atPause (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Pause method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.20 atPrivWrite()

```
ATCA_STATUS atPrivWrite (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand PrivWrite method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.21 atRandom()

```
ATCA_STATUS atRandom (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Random method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.22 atRead()

```
ATCA_STATUS atRead (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Read method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.23 atSecureBoot()

```
ATCA_STATUS atSecureBoot (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand SecureBoot method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.24 atSelfTest()

```
ATCA_STATUS atSelfTest (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.25 atSHA()

```
ATCA_STATUS atSHA (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            uint16_t write_context_size )
```

ATCACommand SHA method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*write_context_size*	the length of the sha write_context data

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.74.2.26 atSign()

```
ATCA_STATUS atSign (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Sign method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.27 atUpdateExtra()

```
ATCA_STATUS atUpdateExtra (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand UpdateExtra method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.74.2.28 atVerify()

```
ATCA_STATUS atVerify (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand ECDSA Verify method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.74.2.29 atWrite()

```
ATCA_STATUS atWrite (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand Write method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	Flag to indicate whether a mac is present or not

Returns

ATCA_SUCCESS

10.74.2.30 isATCAError()

```
ATCA_STATUS isATCAError (
            uint8_t * data )
```

checks for basic error frame in data

Parameters

in	*data*	pointer to received data - expected to be in the form of a CA device response frame

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.75 calib_command.h File Reference

Microchip Crypto Auth device command object - this is a command builder only, it does not send the command. The result of a command method is a fully formed packet, ready to send to the ATCAIFace object to dispatch.

```
#include <stddef.h>
```

Data Structures

- struct ATCAPacket

Macros

- #define ATCA_CMD_SIZE_MIN ((uint8_t)7)
  minimum number of bytes in command (from count byte to second CRC byte)
- #define ATCA_CMD_SIZE_MAX ((uint8_t)4 ∗ 36 + 7)
  maximum size of command packet (Verify)
- #define CMD_STATUS_SUCCESS ((uint8_t)0x00)
  status byte for success
- #define CMD_STATUS_WAKEUP ((uint8_t)0x11)
  status byte after wake-up
- #define CMD_STATUS_BYTE_PARSE ((uint8_t)0x03)
  command parse error
- #define CMD_STATUS_BYTE_ECC ((uint8_t)0x05)
  command ECC error
- #define CMD_STATUS_BYTE_EXEC ((uint8_t)0x0F)

command execution error
- #define CMD_STATUS_BYTE_COMM ((uint8_t)0xFF)

  communication error

Opcodes for Crypto Authentication device commands

- #define ATCA_CHECKMAC ((uint8_t)0x28)

  CheckMac command op-code.
- #define ATCA_DERIVE_KEY ((uint8_t)0x1C)

  DeriveKey command op-code.
- #define ATCA_INFO ((uint8_t)0x30)

  Info command op-code.
- #define ATCA_GENDIG ((uint8_t)0x15)

  GenDig command op-code.
- #define ATCA_GENKEY ((uint8_t)0x40)

  GenKey command op-code.
- #define ATCA_HMAC ((uint8_t)0x11)

  HMAC command op-code.
- #define ATCA_LOCK ((uint8_t)0x17)

  Lock command op-code.
- #define ATCA_MAC ((uint8_t)0x08)

  MAC command op-code.
- #define ATCA_NONCE ((uint8_t)0x16)

  Nonce command op-code.
- #define ATCA_PAUSE ((uint8_t)0x01)

  Pause command op-code.
- #define ATCA_PRIVWRITE ((uint8_t)0x46)

  PrivWrite command op-code.
- #define ATCA_RANDOM ((uint8_t)0x1B)

  Random command op-code.
- #define ATCA_READ ((uint8_t)0x02)

  Read command op-code.
- #define ATCA_SIGN ((uint8_t)0x41)

  Sign command op-code.
- #define ATCA_UPDATE_EXTRA ((uint8_t)0x20)

  UpdateExtra command op-code.
- #define ATCA_VERIFY ((uint8_t)0x45)

  GenKey command op-code.
- #define ATCA_WRITE ((uint8_t)0x12)

  Write command op-code.
- #define ATCA_ECDH ((uint8_t)0x43)

  ECDH command op-code.
- #define ATCA_COUNTER ((uint8_t)0x24)

  Counter command op-code.
- #define ATCA_SHA ((uint8_t)0x47)

  SHA command op-code.
- #define ATCA_AES ((uint8_t)0x51)

  AES command op-code.
- #define ATCA_KDF ((uint8_t)0x56)

  KDF command op-code.
- #define ATCA_SECUREBOOT ((uint8_t)0x80)

  Secure Boot command op-code.
- #define ATCA_SELFTEST ((uint8_t)0x77)

  Self test command op-code.

Definitions of Data and Packet Sizes

- #define ATCA_BLOCK_SIZE (32)

size of a block
- #define ATCA_WORD_SIZE (4)

  size of a word
- #define ATCA_PUB_KEY_PAD (4)

  size of the public key pad
- #define ATCA_SERIAL_NUM_SIZE (9)

  number of bytes in the device serial number
- #define ATCA_RSP_SIZE_VAL ((uint8_t)7)

  size of response packet containing four bytes of data
- #define ATCA_KEY_COUNT (16)

  number of keys
- #define ATCA_ECC_CONFIG_SIZE (128)

  size of configuration zone
- #define ATCA_SHA_CONFIG_SIZE (88)

  size of configuration zone
- #define ATCA_ECC204_CONFIG_SIZE (64)

  size of ECC204 configuration zone
- #define ATCA_ECC204_CONFIG_SLOT_SIZE (16)

  size of ECC204 configuration slot size
- #define ATCA_OTP_SIZE (64)

  size of OTP zone
- #define ATCA_DATA_SIZE (ATCA_KEY_COUNT * ATCA_KEY_SIZE)

  size of data zone
- #define ATCA_AES_GFM_SIZE ATCA_BLOCK_SIZE

  size of GFM data
- #define ATCA_CHIPMODE_OFFSET (19)

  ChipMode byte offset within the configuration zone.
- #define ATCA_CHIPMODE_I2C_ADDRESS_FLAG ((uint8_t)0x01)

  ChipMode I2C Address in UserExtraAdd flag.
- #define ATCA_CHIPMODE_TTL_ENABLE_FLAG ((uint8_t)0x02)

  ChipMode TTLenable flag.
- #define ATCA_CHIPMODE_WATCHDOG_MASK ((uint8_t)0x04)

  ChipMode watchdog duration mask.
- #define ATCA_CHIPMODE_WATCHDOG_SHORT ((uint8_t)0x00)

  ChipMode short watchdog (∼1.3s)
- #define ATCA_CHIPMODE_WATCHDOG_LONG ((uint8_t)0x04)

  ChipMode long watchdog (∼13s)
- #define ATCA_CHIPMODE_CLOCK_DIV_MASK ((uint8_t)0xF8)

  ChipMode clock divider mask.
- #define ATCA_CHIPMODE_CLOCK_DIV_M0 ((uint8_t)0x00)

  ChipMode clock divider M0.
- #define ATCA_CHIPMODE_CLOCK_DIV_M1 ((uint8_t)0x28)

  ChipMode clock divider M1.
- #define ATCA_CHIPMODE_CLOCK_DIV_M2 ((uint8_t)0x68)

  ChipMode clock divider M2.
- #define ATCA_COUNT_SIZE ((uint8_t)1)

  Number of bytes in the command packet Count.
- #define ATCA_CRC_SIZE ((uint8_t)2)

  Number of bytes in the command packet CRC.
- #define ATCA_PACKET_OVERHEAD (ATCA_COUNT_SIZE + ATCA_CRC_SIZE)

  Number of bytes in the command packet.
- #define ATCA_PUB_KEY_SIZE (64)

  size of a p256 public key
- #define ATCA_PRIV_KEY_SIZE (32)

  size of a p256 private key
- #define ATCA_SIG_SIZE (64)

  size of a p256 signature
- #define ATCA_KEY_SIZE (32)

  size of a symmetric SHA key

- #define RSA2048_KEY_SIZE (256)
  size of a RSA private key
- #define ATCA_RSP_SIZE_MIN ((uint8_t)4)
  minimum number of bytes in response
- #define ATCA_RSP_SIZE_4 ((uint8_t)7)
  size of response packet containing 4 bytes data
- #define ATCA_RSP_SIZE_72 ((uint8_t)75)
  size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_64 ((uint8_t)67)
  size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_32 ((uint8_t)35)
  size of response packet containing 32 bytes data
- #define ATCA_RSP_SIZE_16 ((uint8_t)19)
  size of response packet containing 16 bytes data
- #define ATCA_RSP_SIZE_MAX ((uint8_t)75)
  maximum size of response packet (GenKey and Verify command)
- #define OUTNONCE_SIZE (32)
  Size of the OutNonce response expected from several commands.

Definitions for Command Parameter Ranges

- #define ATCA_KEY_ID_MAX ((uint8_t)15)
  maximum value for key id
- #define ATCA_OTP_BLOCK_MAX ((uint8_t)1)
  maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define ATCA_COUNT_IDX (0)
  command packet index for count
- #define ATCA_OPCODE_IDX (1)
  command packet index for op-code
- #define ATCA_PARAM1_IDX (2)
  command packet index for first parameter
- #define ATCA_PARAM2_IDX (3)
  command packet index for second parameter
- #define ATCA_DATA_IDX (5)
  command packet index for data load
- #define ATCA_RSP_DATA_IDX (1)
  buffer index of data in response

Definitions for Zone and Address Parameters

- #define ATCA_ZONE_MASK ((uint8_t)0x03)
  Zone mask.
- #define ATCA_ZONE_ENCRYPTED ((uint8_t)0x40)
  Zone bit 6 set: Write is encrypted with an unlocked data zone.
- #define ATCA_ZONE_READWRITE_32 ((uint8_t)0x80)
  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define ATCA_ADDRESS_MASK_CONFIG (0x001F)
  Address bits 5 to 7 are 0 for Configuration zone.
- #define ATCA_ADDRESS_MASK_OTP (0x000F)
  Address bits 4 to 7 are 0 for OTP zone.
- #define ATCA_ADDRESS_MASK (0x007F)
  Address bit 7 to 15 are always 0.
- #define ATCA_TEMPKEY_KEYID (0xFFFF)
  KeyID when referencing TempKey.

Definitions for Key types

- #define ATCA_B283_KEY_TYPE 0
  B283 NIST ECC key.
- #define ATCA_K283_KEY_TYPE 1
  K283 NIST ECC key.
- #define ATCA_P256_KEY_TYPE 4
  P256 NIST ECC key.
- #define ATCA_AES_KEY_TYPE 6
  AES-128 Key.
- #define ATCA_SHA_KEY_TYPE 7
  SHA key or other data.

Definitions for the AES Command

- #define AES_MODE_IDX ATCA_PARAM1_IDX
  AES command index for mode.
- #define AES_KEYID_IDX ATCA_PARAM2_IDX
  AES command index for key id.
- #define AES_INPUT_IDX ATCA_DATA_IDX
  AES command index for input data.
- #define AES_COUNT (23)
  AES command packet size.
- #define AES_MODE_MASK ((uint8_t)0xC7)
  AES mode bits 3 to 5 are 0.
- #define AES_MODE_KEY_BLOCK_MASK ((uint8_t)0xC0)
  AES mode mask for key block field.
- #define AES_MODE_OP_MASK ((uint8_t)0x07)
  AES mode operation mask.
- #define AES_MODE_ENCRYPT ((uint8_t)0x00)
  AES mode: Encrypt.
- #define AES_MODE_DECRYPT ((uint8_t)0x01)
  AES mode: Decrypt.
- #define AES_MODE_GFM ((uint8_t)0x03)
  AES mode: GFM calculation.
- #define AES_MODE_KEY_BLOCK_POS (6)
  Bit shift for key block in mode.
- #define AES_DATA_SIZE (16)
  size of AES encrypt/decrypt data
- #define AES_RSP_SIZE ATCA_RSP_SIZE_16
  AES command response packet size.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX ATCA_PARAM1_IDX
  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX ATCA_PARAM2_IDX
  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX ATCA_DATA_IDX
  CheckMAC command index for client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_IDX (37)
  CheckMAC command index for client response.
- #define CHECKMAC_DATA_IDX (69)
  CheckMAC command index for other data.
- #define CHECKMAC_COUNT (84)
  CheckMAC command packet size.
- #define CHECKMAC_MODE_CHALLENGE ((uint8_t)0x00)
  CheckMAC mode 0: first SHA block from key id.

- #define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
  CheckMAC mode bit 0: second SHA block from TempKey.
- #define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
  CheckMAC mode bit 1: first SHA block from TempKey.
- #define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
  CheckMAC mode bit 2: match TempKey.SourceFlag.
- #define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
  CheckMAC mode bit 5: include first 64 OTP bits.
- #define CHECKMAC_MODE_MASK ((uint8_t)0x27)
  CheckMAC mode bits 3, 4, 6, and 7 are 0.
- #define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)
  CheckMAC size of client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_SIZE (32)
  CheckMAC size of client response.
- #define CHECKMAC_OTHER_DATA_SIZE (13)
  CheckMAC size of "other data".
- #define CHECKMAC_CLIENT_COMMAND_SIZE (4)
  CheckMAC size of client command header size inside "other data".
- #define CHECKMAC_CMD_MATCH (0)
  CheckMAC return value when there is a match.
- #define CHECKMAC_CMD_MISMATCH (1)
  CheckMAC return value when there is a mismatch.
- #define CHECKMAC_RSP_SIZE ATCA_RSP_SIZE_MIN
  CheckMAC response packet size.

Definitions for the Counter command

- #define COUNTER_COUNT ATCA_CMD_SIZE_MIN
- #define COUNTER_MODE_IDX ATCA_PARAM1_IDX
  Counter command index for mode.
- #define COUNTER_KEYID_IDX ATCA_PARAM2_IDX
  Counter command index for key id.
- #define COUNTER_MODE_MASK ((uint8_t)0x01)
  Counter mode bits 1 to 7 are 0.
- #define COUNTER_MAX_VALUE ((uint32_t)2097151)
  Counter maximum value of the counter.
- #define COUNTER_MODE_READ ((uint8_t)0x00)
  Counter command mode for reading.
- #define COUNTER_MODE_INCREMENT ((uint8_t)0x01)
  Counter command mode for incrementing.
- #define COUNTER_RSP_SIZE ATCA_RSP_SIZE_4
  Counter command response packet size.
- #define COUNTER_SIZE ATCA_RSP_SIZE_MIN
  Counter size in binary.

Definitions for the DeriveKey Command

- #define DERIVE_KEY_RANDOM_IDX ATCA_PARAM1_IDX
  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX ATCA_PARAM2_IDX
  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX ATCA_DATA_IDX
  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL ATCA_CMD_SIZE_MIN
  DeriveKey command packet size without MAC.
- #define DERIVE_KEY_MODE ((uint8_t)0x04)
  DeriveKey command mode set to 4 as in datasheet.
- #define DERIVE_KEY_COUNT_LARGE (39)

DeriveKey command packet size with MAC.
- #define DERIVE_KEY_RANDOM_FLAG ((uint8_t)4)

  DeriveKey 1. parameter; has to match TempKey.SourceFlag.
- #define DERIVE_KEY_MAC_SIZE (32)

  DeriveKey MAC size.
- #define DERIVE_KEY_RSP_SIZE ATCA_RSP_SIZE_MIN

  DeriveKey response packet size.

Definitions for the ECDH Command

- #define ECDH_PREFIX_MODE ((uint8_t)0x00)
- #define ECDH_COUNT (ATCA_CMD_SIZE_MIN + ATCA_PUB_KEY_SIZE)
- #define ECDH_MODE_SOURCE_MASK ((uint8_t)0x01)
- #define ECDH_MODE_SOURCE_EEPROM_SLOT ((uint8_t)0x00)
- #define ECDH_MODE_SOURCE_TEMPKEY ((uint8_t)0x01)
- #define ECDH_MODE_OUTPUT_MASK ((uint8_t)0x02)
- #define ECDH_MODE_OUTPUT_CLEAR ((uint8_t)0x00)
- #define ECDH_MODE_OUTPUT_ENC ((uint8_t)0x02)
- #define ECDH_MODE_COPY_MASK ((uint8_t)0x0C)
- #define ECDH_MODE_COPY_COMPATIBLE ((uint8_t)0x00)
- #define ECDH_MODE_COPY_EEPROM_SLOT ((uint8_t)0x04)
- #define ECDH_MODE_COPY_TEMP_KEY ((uint8_t)0x08)
- #define ECDH_MODE_COPY_OUTPUT_BUFFER ((uint8_t)0x0C)
- #define ECDH_KEY_SIZE ATCA_BLOCK_SIZE

  ECDH output data size.
- #define ECDH_RSP_SIZE ATCA_RSP_SIZE_64

  ECDH command packet size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX ATCA_PARAM1_IDX

  GenDig command index for zone.
- #define GENDIG_KEYID_IDX ATCA_PARAM2_IDX

  GenDig command index for key id.
- #define GENDIG_DATA_IDX ATCA_DATA_IDX

  GenDig command index for optional data.
- #define GENDIG_COUNT ATCA_CMD_SIZE_MIN

  GenDig command packet size without "other data".
- #define GENDIG_ZONE_CONFIG ((uint8_t)0)

  GenDig zone id config. Use KeyID to specify any of the four 256-bit blocks of the Configuration zone.
- #define GENDIG_ZONE_OTP ((uint8_t)1)

  GenDig zone id OTP. Use KeyID to specify either the first or second 256-bit block of the OTP zone.
- #define GENDIG_ZONE_DATA ((uint8_t)2)

  GenDig zone id data. Use KeyID to specify a slot in the Data zone or a transport key in the hardware array.
- #define GENDIG_ZONE_SHARED_NONCE ((uint8_t)3)

  GenDig zone id shared nonce. KeyID specifies the location of the input value in the message generation.
- #define GENDIG_ZONE_COUNTER ((uint8_t)4)

  GenDig zone id counter. KeyID specifies the monotonic counter ID to be included in the message generation.
- #define GENDIG_ZONE_KEY_CONFIG ((uint8_t)5)

  GenDig zone id key config. KeyID specifies the slot for which the configuration information is to be included in the message generation.
- #define GENDIG_RSP_SIZE ATCA_RSP_SIZE_MIN

  GenDig command response packet size.

Definitions for the GenKey Command

- #define GENKEY_MODE_IDX ATCA_PARAM1_IDX

  GenKey command index for mode.

- #define GENKEY_KEYID_IDX ATCA_PARAM2_IDX
  GenKey command index for key id.
- #define GENKEY_DATA_IDX (5)
  GenKey command index for other data.
- #define GENKEY_COUNT ATCA_CMD_SIZE_MIN
  GenKey command packet size without "other data".
- #define GENKEY_COUNT_DATA (10)
  GenKey command packet size with "other data".
- #define GENKEY_OTHER_DATA_SIZE (3)
  GenKey size of "other data".
- #define GENKEY_MODE_MASK ((uint8_t)0x1C)
  GenKey mode bits 0 to 1 and 5 to 7 are 0.
- #define GENKEY_MODE_PRIVATE ((uint8_t)0x04)
  GenKey mode: private key generation.
- #define GENKEY_MODE_PUBLIC ((uint8_t)0x00)
  GenKey mode: public key calculation.
- #define GENKEY_MODE_DIGEST ((uint8_t)0x08)
  GenKey mode: PubKey digest will be created after the public key is calculated.
- #define GENKEY_MODE_PUBKEY_DIGEST ((uint8_t)0x10)
  GenKey mode: Calculate PubKey digest on the public key in KeyId.
- #define GENKEY_MODE_MAC ((uint8_t)0x20)
  Genkey mode: Calculate MAC of public key + session key.
- #define GENKEY_PRIVATE_TO_TEMPKEY ((uint16_t)0xFFFF)
  GenKey Create private key and store to tempkey (608 only)
- #define GENKEY_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  GenKey response packet size in Digest mode.
- #define GENKEY_RSP_SIZE_LONG ATCA_RSP_SIZE_64
  GenKey response packet size when returning a public key.

Definitions for the HMAC Command

- #define HMAC_MODE_IDX ATCA_PARAM1_IDX
  HMAC command index for mode.
- #define HMAC_KEYID_IDX ATCA_PARAM2_IDX
  HMAC command index for key id.
- #define HMAC_COUNT ATCA_CMD_SIZE_MIN
  HMAC command packet size.
- #define HMAC_MODE_FLAG_TK_RAND ((uint8_t)0x00)
  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_TK_NORAND ((uint8_t)0x04)
  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_OTP88 ((uint8_t)0x10)
  HMAC mode bit 4: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message.; otherwise, the corresponding message bits are set to zero. Not applicable for ATECC508A.
- #define HMAC_MODE_FLAG_OTP64 ((uint8_t)0x20)
  HMAC mode bit 5: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message.; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored. Not applicable for ATECC508A.
- #define HMAC_MODE_FLAG_FULLSN ((uint8_t)0x40)
  HMAC mode bit 6: If set, include the 48 bits SN[2:3] and SN[4:7] in the message.; otherwise, the corresponding message bits are set to zero.
- #define HMAC_MODE_MASK ((uint8_t)0x74)
  HMAC mode bits 0, 1, 3, and 7 are 0.
- #define HMAC_DIGEST_SIZE (32)
  HMAC size of digest response.
- #define HMAC_RSP_SIZE ATCA_RSP_SIZE_32
  HMAC command response packet size.

Definitions for the Info Command

- #define INFO_PARAM1_IDX ATCA_PARAM1_IDX
  Info command index for 1. parameter.
- #define INFO_PARAM2_IDX ATCA_PARAM2_IDX
  Info command index for 2. parameter.
- #define INFO_COUNT ATCA_CMD_SIZE_MIN
  Info command packet size.
- #define INFO_MODE_REVISION ((uint8_t)0x00)
  Info mode Revision.
- #define INFO_MODE_KEY_VALID ((uint8_t)0x01)
  Info mode KeyValid.
- #define INFO_MODE_STATE ((uint8_t)0x02)
  Info mode State.
- #define INFO_MODE_LOCK_STATUS ((uint8_t)0x02)
  Info mode Lock status for ECC204 device.
- #define INFO_MODE_GPIO ((uint8_t)0x03)
  Info mode GPIO.
- #define INFO_MODE_VOL_KEY_PERMIT ((uint8_t)0x04)
  Info mode GPIO.
- #define INFO_MODE_MAX ((uint8_t)0x03)
  Info mode maximum value.
- #define INFO_NO_STATE ((uint8_t)0x00)
  Info mode is not the state mode.
- #define INFO_OUTPUT_STATE_MASK ((uint8_t)0x01)
  Info output state mask.
- #define INFO_DRIVER_STATE_MASK ((uint8_t)0x02)
  Info driver state mask.
- #define INFO_PARAM2_SET_LATCH_STATE ((uint16_t)0x0002)
  Info param2 to set the persistent latch state.
- #define INFO_PARAM2_LATCH_SET ((uint16_t)0x0001)
  Info param2 to set the persistent latch.
- #define INFO_PARAM2_LATCH_CLEAR ((uint16_t)0x0000)
  Info param2 to clear the persistent latch.
- #define INFO_SIZE ((uint8_t)0x04)
  Info return size.
- #define INFO_RSP_SIZE ATCA_RSP_SIZE_VAL
  Info command response packet size.

Definitions for the KDF Command

- #define KDF_MODE_IDX ATCA_PARAM1_IDX
  KDF command index for mode.
- #define KDF_KEYID_IDX ATCA_PARAM2_IDX
  KDF command index for key id.
- #define KDF_DETAILS_IDX ATCA_DATA_IDX
  KDF command index for details.
- #define KDF_DETAILS_SIZE 4
  KDF details (param3) size.
- #define KDF_MESSAGE_IDX (ATCA_DATA_IDX + KDF_DETAILS_SIZE)
- #define KDF_MODE_SOURCE_MASK ((uint8_t)0x03)
  KDF mode source key mask.
- #define KDF_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
  KDF mode source key in TempKey.
- #define KDF_MODE_SOURCE_TEMPKEY_UP ((uint8_t)0x01)
  KDF mode source key in upper TempKey.
- #define KDF_MODE_SOURCE_SLOT ((uint8_t)0x02)
  KDF mode source key in a slot.
- #define KDF_MODE_SOURCE_ALTKEYBUF ((uint8_t)0x03)

KDF mode source key in alternate key buffer.
- #define KDF_MODE_TARGET_MASK ((uint8_t)0x1C)
  KDF mode target key mask.
- #define KDF_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
  KDF mode target key in TempKey.
- #define KDF_MODE_TARGET_TEMPKEY_UP ((uint8_t)0x04)
  KDF mode target key in upper TempKey.
- #define KDF_MODE_TARGET_SLOT ((uint8_t)0x08)
  KDF mode target key in slot.
- #define KDF_MODE_TARGET_ALTKEYBUF ((uint8_t)0x0C)
  KDF mode target key in alternate key buffer.
- #define KDF_MODE_TARGET_OUTPUT ((uint8_t)0x10)
  KDF mode target key in output buffer.
- #define KDF_MODE_TARGET_OUTPUT_ENC ((uint8_t)0x14)
  KDF mode target key encrypted in output buffer.
- #define KDF_MODE_ALG_MASK ((uint8_t)0x60)
  KDF mode algorithm mask.
- #define KDF_MODE_ALG_PRF ((uint8_t)0x00)
  KDF mode PRF algorithm.
- #define KDF_MODE_ALG_AES ((uint8_t)0x20)
  KDF mode AES algorithm.
- #define KDF_MODE_ALG_HKDF ((uint8_t)0x40)
  KDF mode HKDF algorithm.
- #define KDF_DETAILS_PRF_KEY_LEN_MASK ((uint32_t)0x00000003)
  KDF details for PRF, source key length mask.
- #define KDF_DETAILS_PRF_KEY_LEN_16 ((uint32_t)0x00000000)
  KDF details for PRF, source key length is 16 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_32 ((uint32_t)0x00000001)
  KDF details for PRF, source key length is 32 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_48 ((uint32_t)0x00000002)
  KDF details for PRF, source key length is 48 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_64 ((uint32_t)0x00000003)
  KDF details for PRF, source key length is 64 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_MASK ((uint32_t)0x00000100)
  KDF details for PRF, target length mask.
- #define KDF_DETAILS_PRF_TARGET_LEN_32 ((uint32_t)0x00000000)
  KDF details for PRF, target length is 32 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_64 ((uint32_t)0x00000100)
  KDF details for PRF, target length is 64 bytes.
- #define KDF_DETAILS_PRF_AEAD_MASK ((uint32_t)0x00000600)
  KDF details for PRF, AEAD processing mask.
- #define KDF_DETAILS_PRF_AEAD_MODE0 ((uint32_t)0x00000000)
  KDF details for PRF, AEAD no processing.
- #define KDF_DETAILS_PRF_AEAD_MODE1 ((uint32_t)0x00000200)
  KDF details for PRF, AEAD First 32 go to target, second 32 go to output buffer.
- #define KDF_DETAILS_AES_KEY_LOC_MASK ((uint32_t)0x00000003)
  KDF details for AES, key location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_MASK ((uint32_t)0x00000003)
  KDF details for HKDF, message location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_SLOT ((uint32_t)0x00000000)
  KDF details for HKDF, message location in slot.
- #define KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY ((uint32_t)0x00000001)
  KDF details for HKDF, message location in TempKey.
- #define KDF_DETAILS_HKDF_MSG_LOC_INPUT ((uint32_t)0x00000002)
  KDF details for HKDF, message location in input parameter.
- #define KDF_DETAILS_HKDF_MSG_LOC_IV ((uint32_t)0x00000003)
  KDF details for HKDF, message location is a special IV function.
- #define KDF_DETAILS_HKDF_ZERO_KEY ((uint32_t)0x00000004)
  KDF details for HKDF, key is 32 bytes of zero.

Definitions for the Lock Command

- #define LOCK_ZONE_IDX ATCA_PARAM1_IDX
  Lock command index for zone.
- #define LOCK_SUMMARY_IDX ATCA_PARAM2_IDX
  Lock command index for summary.
- #define LOCK_COUNT ATCA_CMD_SIZE_MIN
  Lock command packet size.
- #define LOCK_ZONE_CONFIG ((uint8_t)0x00)
  Lock zone is Config.
- #define LOCK_ZONE_DATA ((uint8_t)0x01)
  Lock zone is OTP or Data.
- #define LOCK_ZONE_DATA_SLOT ((uint8_t)0x02)
  Lock slot of Data.
- #define LOCK_ECC204_ZONE_DATA ((uint8_t)0x00)
  Lock ECC204 Data zone by slot.
- #define LOCK_ECC204_ZONE_CONFIG ((uint8_t)0x01)
  Lock ECC204 configuration zone by slot.
- #define LOCK_ZONE_NO_CRC ((uint8_t)0x80)
  Lock command: Ignore summary.
- #define LOCK_ZONE_MASK (0xBF)
  Lock parameter 1 bits 6 are 0.
- #define ATCA_UNLOCKED (0x55)
  Value indicating an unlocked zone.
- #define ATCA_LOCKED (0x00)
  Value indicating a locked zone.
- #define LOCK_RSP_SIZE ATCA_RSP_SIZE_MIN
  Lock command response packet size.

Definitions for the MAC Command

- #define MAC_MODE_IDX ATCA_PARAM1_IDX
  MAC command index for mode.
- #define MAC_KEYID_IDX ATCA_PARAM2_IDX
  MAC command index for key id.
- #define MAC_CHALLENGE_IDX ATCA_DATA_IDX
  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT ATCA_CMD_SIZE_MIN
  MAC command packet size without challenge.
- #define MAC_COUNT_LONG (39)
  MAC command packet size with challenge.
- #define MAC_MODE_CHALLENGE ((uint8_t)0x00)
  MAC mode 0: first SHA block from data slot.
- #define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
  MAC mode bit 1: first SHA block from TempKey.
- #define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
  MAC mode bit 2: match TempKey.SourceFlag.
- #define MAC_MODE_PTNONCE_TEMPKEY ((uint8_t)0x06)
  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_PASSTHROUGH ((uint8_t)0x07)
  MAC mode bit 0-2: pass-through mode.
- #define MAC_MODE_INCLUDE_OTP_88 ((uint8_t)0x10)
  MAC mode bit 4: include first 88 OTP bits.
- #define MAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
  MAC mode bit 5: include first 64 OTP bits.
- #define MAC_MODE_INCLUDE_SN ((uint8_t)0x40)
  MAC mode bit 6: include serial number.

- #define MAC_CHALLENGE_SIZE (32)
  MAC size of challenge.
- #define MAC_SIZE (32)
  MAC size of response.
- #define MAC_MODE_MASK ((uint8_t)0x77)
  MAC mode bits 3 and 7 are 0.
- #define MAC_RSP_SIZE ATCA_RSP_SIZE_32
  MAC command response packet size.

Definitions for the Nonce Command

- #define NONCE_MODE_IDX ATCA_PARAM1_IDX
  Nonce command index for mode.
- #define NONCE_PARAM2_IDX ATCA_PARAM2_IDX
  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX ATCA_DATA_IDX
  Nonce command index for input data.
- #define NONCE_COUNT_SHORT (ATCA_CMD_SIZE_MIN + 20)
  Nonce command packet size for 20 bytes of NumIn.
- #define NONCE_COUNT_LONG (ATCA_CMD_SIZE_MIN + 32)
  Nonce command packet size for 32 bytes of NumIn.
- #define NONCE_COUNT_LONG_64 (ATCA_CMD_SIZE_MIN + 64)
  Nonce command packet size for 64 bytes of NumIn.
- #define NONCE_MODE_MASK ((uint8_t)0x03)
  Nonce mode bits 2 to 7 are 0.
- #define NONCE_MODE_SEED_UPDATE ((uint8_t)0x00)
  Nonce mode: update seed.
- #define NONCE_MODE_NO_SEED_UPDATE ((uint8_t)0x01)
  Nonce mode: do not update seed.
- #define NONCE_MODE_INVALID ((uint8_t)0x02)
  Nonce mode 2 is invalid.
- #define NONCE_MODE_PASSTHROUGH ((uint8_t)0x03)
  Nonce mode: pass-through.
- #define NONCE_MODE_GEN_SESSION_KEY ((uint8_t)0x02)
  NOnce mode: Generate session key in ECC204 device.
- #define NONCE_MODE_INPUT_LEN_MASK ((uint8_t)0x20)
  Nonce mode: input size mask.
- #define NONCE_MODE_INPUT_LEN_32 ((uint8_t)0x00)
  Nonce mode: input size is 32 bytes.
- #define NONCE_MODE_INPUT_LEN_64 ((uint8_t)0x20)
  Nonce mode: input size is 64 bytes.
- #define NONCE_MODE_TARGET_MASK ((uint8_t)0xC0)
  Nonce mode: target mask.
- #define NONCE_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
  Nonce mode: target is TempKey.
- #define NONCE_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
  Nonce mode: target is Message Digest Buffer.
- #define NONCE_MODE_TARGET_ALTKEYBUF ((uint8_t)0x80)
  Nonce mode: target is Alternate Key Buffer.
- #define NONCE_ZERO_CALC_MASK ((uint16_t)0x8000)
  Nonce zero (param2): calculation mode mask.
- #define NONCE_ZERO_CALC_RANDOM ((uint16_t)0x0000)
  Nonce zero (param2): calculation mode random, use RNG in calculation and return RNG output.
- #define NONCE_ZERO_CALC_TEMPKEY ((uint16_t)0x8000)
  Nonce zero (param2): calculation mode TempKey, use TempKey in calculation and return new TempKey value.
- #define NONCE_NUMIN_SIZE (20)
  Nonce NumIn size for random modes.
- #define NONCE_NUMIN_SIZE_PASSTHROUGH (32)

Nonce NumIn size for 32-byte pass-through mode.
- #define NONCE_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  Nonce command response packet size with no output.
- #define NONCE_RSP_SIZE_LONG ATCA_RSP_SIZE_32
  Nonce command response packet size with output.

Definitions for the Pause Command

- #define PAUSE_SELECT_IDX ATCA_PARAM1_IDX
  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX ATCA_PARAM2_IDX
  Pause command index for 2. parameter.
- #define PAUSE_COUNT ATCA_CMD_SIZE_MIN
  Pause command packet size.
- #define PAUSE_RSP_SIZE ATCA_RSP_SIZE_MIN
  Pause command response packet size.

Definitions for the PrivWrite Command

- #define PRIVWRITE_ZONE_IDX ATCA_PARAM1_IDX
  PrivWrite command index for zone.
- #define PRIVWRITE_KEYID_IDX ATCA_PARAM2_IDX
  PrivWrite command index for KeyID.
- #define PRIVWRITE_VALUE_IDX (5)
  PrivWrite command index for value.
- #define PRIVWRITE_MAC_IDX (41)
  PrivWrite command index for MAC.
- #define PRIVWRITE_COUNT (75)
  PrivWrite command packet size.
- #define PRIVWRITE_ZONE_MASK ((uint8_t)0x40)
  PrivWrite zone bits 0 to 5 and 7 are 0.
- #define PRIVWRITE_MODE_ENCRYPT ((uint8_t)0x40)
  PrivWrite mode: encrypted.
- #define PRIVWRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
  PrivWrite command response packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX ATCA_PARAM1_IDX
  Random command index for mode.
- #define RANDOM_PARAM2_IDX ATCA_PARAM2_IDX
  Random command index for 2. parameter.
- #define RANDOM_COUNT ATCA_CMD_SIZE_MIN
  Random command packet size.
- #define RANDOM_SEED_UPDATE ((uint8_t)0x00)
  Random mode for automatic seed update.
- #define RANDOM_NO_SEED_UPDATE ((uint8_t)0x01)
  Random mode for no seed update.
- #define RANDOM_NUM_SIZE ((uint8_t)32)
  Number of bytes in the data packet of a random command.
- #define RANDOM_RSP_SIZE ATCA_RSP_SIZE_32
  Random command response packet size.

Definitions for the Read Command

- #define READ_ZONE_IDX ATCA_PARAM1_IDX

Read command index for zone.
- #define READ_ADDR_IDX ATCA_PARAM2_IDX

  Read command index for address.
- #define READ_COUNT ATCA_CMD_SIZE_MIN

  Read command packet size.
- #define READ_ZONE_MASK ((uint8_t)0x83)

  Read zone bits 2 to 6 are 0.
- #define READ_4_RSP_SIZE ATCA_RSP_SIZE_VAL

  Read command response packet size when reading 4 bytes.
- #define READ_32_RSP_SIZE ATCA_RSP_SIZE_32

  Read command response packet size when reading 32 bytes.

Definitions for the SecureBoot Command

- #define SECUREBOOT_MODE_IDX ATCA_PARAM1_IDX

  SecureBoot command index for mode.
- #define SECUREBOOT_DIGEST_SIZE (32)

  SecureBoot digest input size.
- #define SECUREBOOT_SIGNATURE_SIZE (64)

  SecureBoot signature input size.
- #define SECUREBOOT_COUNT_DIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE)

  SecureBoot command packet size for just a digest.
- #define SECUREBOOT_COUNT_DIG_SIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE + SECUREBOOT_SIGNATURE_SIZE)

  SecureBoot command packet size for a digest and signature.
- #define SECUREBOOT_MAC_SIZE (32)

  SecureBoot MAC output size.
- #define SECUREBOOT_RSP_SIZE_NO_MAC ATCA_RSP_SIZE_MIN

  SecureBoot response packet size for no MAC.
- #define SECUREBOOT_RSP_SIZE_MAC (ATCA_PACKET_OVERHEAD + SECUREBOOT_MAC_SIZE)

  SecureBoot response packet size with MAC.
- #define SECUREBOOT_MODE_MASK ((uint8_t)0x07)

  SecureBoot mode mask.
- #define SECUREBOOT_MODE_FULL ((uint8_t)0x05)

  SecureBoot mode Full.
- #define SECUREBOOT_MODE_FULL_STORE ((uint8_t)0x06)

  SecureBoot mode FullStore.
- #define SECUREBOOT_MODE_FULL_COPY ((uint8_t)0x07)

  SecureBoot mode FullCopy.
- #define SECUREBOOT_MODE_PROHIBIT_FLAG ((uint8_t)0x40)

  SecureBoot mode flag to prohibit SecureBoot until next power cycle.
- #define SECUREBOOT_MODE_ENC_MAC_FLAG ((uint8_t)0x80)

  SecureBoot mode flag for encrypted digest and returning validating MAC.
- #define SECUREBOOTCONFIG_OFFSET (70)

  SecureBootConfig byte offset into the configuration zone.
- #define SECUREBOOTCONFIG_MODE_MASK ((uint16_t)0x0003)

  Mask for SecureBootMode field in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_DISABLED ((uint16_t)0x0000)

  Disabled SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_BOTH ((uint16_t)0x0001)

  Both digest and signature always required SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_SIG ((uint16_t)0x0002)

  Signature stored SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_DIG ((uint16_t)0x0003)

  Digest stored SecureBootMode in SecureBootConfig value.

Definitions for the SelfTest Command

- #define SELFTEST_MODE_IDX ATCA_PARAM1_IDX
  SelfTest command index for mode.
- #define SELFTEST_COUNT ATCA_CMD_SIZE_MIN
  SelfTest command packet size.
- #define SELFTEST_MODE_RNG ((uint8_t)0x01)
  SelfTest mode RNG DRBG function.
- #define SELFTEST_MODE_ECDSA_SIGN_VERIFY ((uint8_t)0x02)
  SelfTest mode ECDSA verify function.
- #define SELFTEST_MODE_ECDH ((uint8_t)0x08)
  SelfTest mode ECDH function.
- #define SELFTEST_MODE_AES ((uint8_t)0x10)
  SelfTest mode AES encrypt function.
- #define SELFTEST_MODE_SHA ((uint8_t)0x20)
  SelfTest mode SHA function.
- #define SELFTEST_MODE_ALL ((uint8_t)0x3B)
  SelfTest mode all algorithms.
- #define SELFTEST_RSP_SIZE ATCA_RSP_SIZE_MIN
  SelfTest command response packet size.

Definitions for the SHA Command

- #define SHA_COUNT_SHORT ATCA_CMD_SIZE_MIN
- #define SHA_COUNT_LONG ATCA_CMD_SIZE_MIN
  Just a starting size.
- #define ATCA_SHA_DIGEST_SIZE (32)
- #define SHA_DATA_MAX (64)
- #define SHA_MODE_MASK ((uint8_t)0x07)
  Mask the bit 0-2.
- #define SHA_MODE_SHA256_START ((uint8_t)0x00)
  Initialization, does not accept a message.
- #define SHA_MODE_SHA256_UPDATE ((uint8_t)0x01)
  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_SHA256_END ((uint8_t)0x02)
  Complete the calculation and return the digest.
- #define SHA_MODE_SHA256_PUBLIC ((uint8_t)0x03)
  Add 64 byte ECC public key in the slot to the SHA context.
- #define SHA_MODE_HMAC_START ((uint8_t)0x04)
  Initialization, HMAC calculation.
- #define SHA_MODE_ECC204_HMAC_START ((uint8_t)0x03)
  Initialization, HMAC calculation for ECC204.
- #define SHA_MODE_HMAC_UPDATE ((uint8_t)0x01)
  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_HMAC_END ((uint8_t)0x05)
  Complete the HMAC computation and return digest.
- #define SHA_MODE_608_HMAC_END ((uint8_t)0x02)
  Complete the HMAC computation and return digest... Different command on 608.
- #define SHA_MODE_ECC204_HMAC_END ((uint8_t)0x02)
  Complete the HMAC computation and return digest... Different mode on ECC204.
- #define SHA_MODE_READ_CONTEXT ((uint8_t)0x06)
  Read current SHA-256 context out of the device.
- #define SHA_MODE_WRITE_CONTEXT ((uint8_t)0x07)
  Restore a SHA-256 context into the device.
- #define SHA_MODE_TARGET_MASK ((uint8_t)0xC0)
  Resulting digest target location mask.
- #define SHA_RSP_SIZE ATCA_RSP_SIZE_32
  SHA command response packet size.
- #define SHA_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  SHA command response packet size only status code.
- #define SHA_RSP_SIZE_LONG ATCA_RSP_SIZE_32

SHA command response packet size.

Definitions for the Sign Command

- #define SIGN_MODE_IDX ATCA_PARAM1_IDX
  Sign command index for mode.
- #define SIGN_KEYID_IDX ATCA_PARAM2_IDX
  Sign command index for key id.
- #define SIGN_COUNT ATCA_CMD_SIZE_MIN
  Sign command packet size.
- #define SIGN_MODE_MASK ((uint8_t)0xE1)
  Sign mode bits 1 to 4 are 0.
- #define SIGN_MODE_INTERNAL ((uint8_t)0x00)
  Sign mode 0: internal.
- #define SIGN_MODE_INVALIDATE ((uint8_t)0x01)
  Sign mode bit 1: Signature will be used for Verify(Invalidate)
- #define SIGN_MODE_INCLUDE_SN ((uint8_t)0x40)
  Sign mode bit 6: include serial number.
- #define SIGN_MODE_EXTERNAL ((uint8_t)0x80)
  Sign mode bit 7: external.
- #define SIGN_MODE_SOURCE_MASK ((uint8_t)0x20)
  Sign mode message source mask.
- #define SIGN_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
  Sign mode message source is TempKey.
- #define SIGN_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
  Sign mode message source is the Message Digest Buffer.
- #define SIGN_RSP_SIZE ATCA_RSP_SIZE_MAX
  Sign command response packet size.

Definitions for the UpdateExtra Command

- #define UPDATE_MODE_IDX ATCA_PARAM1_IDX
  UpdateExtra command index for mode.
- #define UPDATE_VALUE_IDX ATCA_PARAM2_IDX
  UpdateExtra command index for new value.
- #define UPDATE_COUNT ATCA_CMD_SIZE_MIN
  UpdateExtra command packet size.
- #define UPDATE_MODE_USER_EXTRA ((uint8_t)0x00)
  UpdateExtra mode update UserExtra (config byte 84)
- #define UPDATE_MODE_SELECTOR ((uint8_t)0x01)
  UpdateExtra mode update Selector (config byte 85)
- #define UPDATE_MODE_USER_EXTRA_ADD UPDATE_MODE_SELECTOR
  UpdateExtra mode update UserExtraAdd (config byte 85)
- #define UPDATE_MODE_DEC_COUNTER ((uint8_t)0x02)
  UpdateExtra mode: decrement counter.
- #define UPDATE_RSP_SIZE ATCA_RSP_SIZE_MIN
  UpdateExtra command response packet size.

Definitions for the Verify Command

- #define VERIFY_MODE_IDX ATCA_PARAM1_IDX
  Verify command index for mode.
- #define VERIFY_KEYID_IDX ATCA_PARAM2_IDX
  Verify command index for key id.
- #define VERIFY_DATA_IDX (5)
  Verify command index for data.
- #define VERIFY_256_STORED_COUNT (71)

Verify command packet size for 256-bit key in stored mode.
- #define VERIFY_283_STORED_COUNT (79)

  Verify command packet size for 283-bit key in stored mode.
- #define VERIFY_256_VALIDATE_COUNT (90)

  Verify command packet size for 256-bit key in validate mode.
- #define VERIFY_283_VALIDATE_COUNT (98)

  Verify command packet size for 283-bit key in validate mode.
- #define VERIFY_256_EXTERNAL_COUNT (135)

  Verify command packet size for 256-bit key in external mode.
- #define VERIFY_283_EXTERNAL_COUNT (151)

  Verify command packet size for 283-bit key in external mode.
- #define VERIFY_256_KEY_SIZE (64)

  Verify key size for 256-bit key.
- #define VERIFY_283_KEY_SIZE (72)

  Verify key size for 283-bit key.
- #define VERIFY_256_SIGNATURE_SIZE (64)

  Verify signature size for 256-bit key.
- #define VERIFY_283_SIGNATURE_SIZE (72)

  Verify signature size for 283-bit key.
- #define VERIFY_OTHER_DATA_SIZE (19)

  Verify size of "other data".
- #define VERIFY_MODE_MASK ((uint8_t)0x07)

  Verify mode bits 3 to 7 are 0.
- #define VERIFY_MODE_STORED ((uint8_t)0x00)

  Verify mode: stored.
- #define VERIFY_MODE_VALIDATE_EXTERNAL ((uint8_t)0x01)

  Verify mode: validate external.
- #define VERIFY_MODE_EXTERNAL ((uint8_t)0x02)

  Verify mode: external.
- #define VERIFY_MODE_VALIDATE ((uint8_t)0x03)

  Verify mode: validate.
- #define VERIFY_MODE_INVALIDATE ((uint8_t)0x07)

  Verify mode: invalidate.
- #define VERIFY_MODE_SOURCE_MASK ((uint8_t)0x20)

  Verify mode message source mask.
- #define VERIFY_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)

  Verify mode message source is TempKey.
- #define VERIFY_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)

  Verify mode message source is the Message Digest Buffer.
- #define VERIFY_MODE_MAC_FLAG ((uint8_t)0x80)

  Verify mode: MAC.
- #define VERIFY_KEY_B283 ((uint16_t)0x0000)

  Verify key type: B283.
- #define VERIFY_KEY_K283 ((uint16_t)0x0001)

  Verify key type: K283.
- #define VERIFY_KEY_P256 ((uint16_t)0x0004)

  Verify key type: P256.
- #define VERIFY_RSP_SIZE ATCA_RSP_SIZE_MIN

  Verify command response packet size.
- #define VERIFY_RSP_SIZE_MAC ATCA_RSP_SIZE_32

  Verify command response packet size with validating MAC.

Definitions for the Write Command

- #define WRITE_ZONE_IDX ATCA_PARAM1_IDX

  Write command index for zone.
- #define WRITE_ADDR_IDX ATCA_PARAM2_IDX

  Write command index for address.

- #define WRITE_VALUE_IDX ATCA_DATA_IDX
  Write command index for data.
- #define WRITE_MAC_VS_IDX (9)
  Write command index for MAC following short data.
- #define WRITE_MAC_VL_IDX (37)
  Write command index for MAC following long data.
- #define WRITE_MAC_SIZE (32)
  Write MAC size.
- #define WRITE_ZONE_MASK ((uint8_t)0xC3)
  Write zone bits 2 to 5 are 0.
- #define WRITE_ZONE_WITH_MAC ((uint8_t)0x40)
  Write zone bit 6: write encrypted with MAC.
- #define WRITE_ZONE_OTP ((uint8_t)1)
  Write zone id OTP.
- #define WRITE_ZONE_DATA ((uint8_t)2)
  Write zone id data.
- #define WRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
  Write command response packet size.

Functions

- ATCA_STATUS atCheckMAC (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand CheckMAC method.
- ATCA_STATUS atCounter (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Counter method.
- ATCA_STATUS atDeriveKey (ATCADeviceType device_type, ATCAPacket ∗packet, bool has_mac)
  ATCACommand DeriveKey method.
- ATCA_STATUS atECDH (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand ECDH method.
- ATCA_STATUS atGenDig (ATCADeviceType device_type, ATCAPacket ∗packet, bool is_no_mac_key)
  ATCACommand Generate Digest method.
- ATCA_STATUS atGenKey (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Generate Key method.
- ATCA_STATUS atHMAC (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand HMAC method.
- ATCA_STATUS atInfo (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Info method.
- ATCA_STATUS atLock (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Lock method.
- ATCA_STATUS atMAC (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand MAC method.
- ATCA_STATUS atNonce (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Nonce method.
- ATCA_STATUS atPause (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Pause method.
- ATCA_STATUS atPrivWrite (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand PrivWrite method.
- ATCA_STATUS atRandom (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Random method.
- ATCA_STATUS atRead (ATCADeviceType device_type, ATCAPacket ∗packet)
  ATCACommand Read method.
- ATCA_STATUS atSecureBoot (ATCADeviceType device_type, ATCAPacket ∗packet)

ATCACommand SecureBoot method.

- ATCA_STATUS atSHA (ATCADeviceType device_type, ATCAPacket ∗packet, uint16_t write_context_size)

  ATCACommand SHA method.
- ATCA_STATUS atSign (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand Sign method.
- ATCA_STATUS atUpdateExtra (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand UpdateExtra method.
- ATCA_STATUS atVerify (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand ECDSA Verify method.
- ATCA_STATUS atWrite (ATCADeviceType device_type, ATCAPacket ∗packet, bool has_mac)

  ATCACommand Write method.
- ATCA_STATUS atAES (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atSelfTest (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atKDF (ATCADeviceType device_type, ATCAPacket ∗packet)

  ATCACommand KDF method.
- bool atIsSHAFamily (ATCADeviceType device_type)

  determines if a given device type is a SHA device or a superset of a SHA device
- bool atIsECCFamily (ATCADeviceType device_type)

  determines if a given device type is an ECC device or a superset of a ECC device
- ATCA_STATUS isATCAError (uint8_t ∗data)

  checks for basic error frame in data
- void atCRC (size_t length, const uint8_t ∗data, uint8_t ∗crc_le)

  Calculates CRC over the given raw data and returns the CRC in little-endian byte order.
- void atCalcCrc (ATCAPacket ∗pkt)

  This function calculates CRC and adds it to the correct offset in the packet data.
- ATCA_STATUS atCheckCrc (const uint8_t ∗response)

  This function checks the consistency of a response.

10.75.1 Detailed Description

Microchip Crypto Auth device command object - this is a command builder only, it does not send the command. The result of a command method is a fully formed packet, ready to send to the ATCAIFace object to dispatch.

This command object supports the ATSHA and ATECC device family. The command list is a superset of all device commands for this family. The command object differentiates the packet contents based on specific device type within the family.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.75.2 Macro Definition Documentation

10.75.2.1 AES_COUNT

```
#define AES_COUNT (23)
```

AES command packet size.

10.75.2.2 AES_DATA_SIZE

```
#define AES_DATA_SIZE (16)
```

size of AES encrypt/decrypt data

10.75.2.3 AES_INPUT_IDX

```
#define AES_INPUT_IDX ATCA_DATA_IDX
```

AES command index for input data.

10.75.2.4 AES_KEYID_IDX

```
#define AES_KEYID_IDX ATCA_PARAM2_IDX
```

AES command index for key id.

10.75.2.5 AES_MODE_DECRYPT

```
#define AES_MODE_DECRYPT ((uint8_t)0x01)
```

AES mode: Decrypt.

10.75.2.6 AES_MODE_ENCRYPT

```
#define AES_MODE_ENCRYPT ((uint8_t)0x00)
```

AES mode: Encrypt.

10.75.2.7 AES_MODE_GFM

```
#define AES_MODE_GFM ((uint8_t)0x03)
```

AES mode: GFM calculation.

10.75.2.8 AES_MODE_IDX

```
#define AES_MODE_IDX ATCA_PARAM1_IDX
```

AES command index for mode.

10.75.2.9 AES_MODE_KEY_BLOCK_MASK

```
#define AES_MODE_KEY_BLOCK_MASK ((uint8_t)0xC0)
```

AES mode mask for key block field.

10.75.2.10 AES_MODE_KEY_BLOCK_POS

```
#define AES_MODE_KEY_BLOCK_POS (6)
```

Bit shift for key block in mode.

10.75.2.11 AES_MODE_MASK

```
#define AES_MODE_MASK ((uint8_t)0xC7)
```

AES mode bits 3 to 5 are 0.

10.75.2.12 AES_MODE_OP_MASK

```
#define AES_MODE_OP_MASK ((uint8_t)0x07)
```

AES mode operation mask.

10.75.2.13 AES_RSP_SIZE

```
#define AES_RSP_SIZE ATCA_RSP_SIZE_16
```

AES command response packet size.

10.75.2.14 ATCA_ADDRESS_MASK

```
#define ATCA_ADDRESS_MASK (0x007F)
```

Address bit 7 to 15 are always 0.

10.75.2.15 ATCA_ADDRESS_MASK_CONFIG

```
#define ATCA_ADDRESS_MASK_CONFIG (0x001F)
```

Address bits 5 to 7 are 0 for Configuration zone.

10.75.2.16 ATCA_ADDRESS_MASK_OTP

```
#define ATCA_ADDRESS_MASK_OTP (0x000F)
```

Address bits 4 to 7 are 0 for OTP zone.

10.75.2.17 ATCA_AES

```
#define ATCA_AES ((uint8_t)0x51)
```

AES command op-code.

10.75.2.18 ATCA_AES_GFM_SIZE

```
#define ATCA_AES_GFM_SIZE ATCA_BLOCK_SIZE
```

size of GFM data

10.75.2.19 ATCA_AES_KEY_TYPE

```
#define ATCA_AES_KEY_TYPE 6
```

AES-128 Key.

10.75.2.20 ATCA_B283_KEY_TYPE

```
#define ATCA_B283_KEY_TYPE 0
```

B283 NIST ECC key.

10.75.2.21 ATCA_BLOCK_SIZE

```
#define ATCA_BLOCK_SIZE (32)
```

size of a block

10.75.2.22 ATCA_CHECKMAC

```
#define ATCA_CHECKMAC ((uint8_t)0x28)
```

CheckMac command op-code.

10.75.2.23 ATCA_CHIPMODE_CLOCK_DIV_M0

```
#define ATCA_CHIPMODE_CLOCK_DIV_M0 ((uint8_t)0x00)
```

ChipMode clock divider M0.

10.75.2.24 ATCA_CHIPMODE_CLOCK_DIV_M1

```
#define ATCA_CHIPMODE_CLOCK_DIV_M1 ((uint8_t)0x28)
```

ChipMode clock divider M1.

10.75.2.25 ATCA_CHIPMODE_CLOCK_DIV_M2

```
#define ATCA_CHIPMODE_CLOCK_DIV_M2 ((uint8_t)0x68)
```

ChipMode clock divider M2.

10.75.2.26 ATCA_CHIPMODE_CLOCK_DIV_MASK

```
#define ATCA_CHIPMODE_CLOCK_DIV_MASK ((uint8_t)0xF8)
```

ChipMode clock divider mask.

10.75.2.27 ATCA_CHIPMODE_I2C_ADDRESS_FLAG

```
#define ATCA_CHIPMODE_I2C_ADDRESS_FLAG ((uint8_t)0x01)
```

ChipMode I2C Address in UserExtraAdd flag.

10.75.2.28 ATCA_CHIPMODE_OFFSET

```
#define ATCA_CHIPMODE_OFFSET (19)
```

ChipMode byte offset within the configuration zone.

10.75.2.29 ATCA_CHIPMODE_TTL_ENABLE_FLAG

```
#define ATCA_CHIPMODE_TTL_ENABLE_FLAG ((uint8_t)0x02)
```

ChipMode TTLenable flag.

10.75.2.30 ATCA_CHIPMODE_WATCHDOG_LONG

```
#define ATCA_CHIPMODE_WATCHDOG_LONG ((uint8_t)0x04)
```

ChipMode long watchdog ($\sim$13s)

10.75.2.31 ATCA_CHIPMODE_WATCHDOG_MASK

```
#define ATCA_CHIPMODE_WATCHDOG_MASK ((uint8_t)0x04)
```

ChipMode watchdog duration mask.

10.75.2.32 ATCA_CHIPMODE_WATCHDOG_SHORT

```
#define ATCA_CHIPMODE_WATCHDOG_SHORT ((uint8_t)0x00)
```

ChipMode short watchdog ($\sim$1.3s)

10.75.2.33 ATCA_CMD_SIZE_MAX

```
#define ATCA_CMD_SIZE_MAX ((uint8_t)4 * 36 + 7)
```

maximum size of command packet (Verify)

10.75.2.34 ATCA_CMD_SIZE_MIN

```
#define ATCA_CMD_SIZE_MIN ((uint8_t)7)
```

minimum number of bytes in command (from count byte to second CRC byte)

10.75.2.35 ATCA_COUNT_IDX

```
#define ATCA_COUNT_IDX (0)
```

command packet index for count

10.75.2.36 ATCA_COUNT_SIZE

```
#define ATCA_COUNT_SIZE ((uint8_t)1)
```

Number of bytes in the command packet Count.

10.75.2.37 ATCA_COUNTER

```
#define ATCA_COUNTER ((uint8_t)0x24)
```

Counter command op-code.

10.75.2.38 ATCA_CRC_SIZE

```
#define ATCA_CRC_SIZE ((uint8_t)2)
```

Number of bytes in the command packet CRC.

10.75.2.39 ATCA_DATA_IDX

```
#define ATCA_DATA_IDX (5)
```

command packet index for data load

10.75.2.40 ATCA_DATA_SIZE

```
#define ATCA_DATA_SIZE (ATCA_KEY_COUNT * ATCA_KEY_SIZE)
```

size of data zone

10.75.2.41 ATCA_DERIVE_KEY

```
#define ATCA_DERIVE_KEY ((uint8_t)0x1C)
```

DeriveKey command op-code.

10.75.2.42 ATCA_ECC204_CONFIG_SIZE

```
#define ATCA_ECC204_CONFIG_SIZE (64)
```

size of ECC204 configuration zone

10.75.2.43 ATCA_ECC204_CONFIG_SLOT_SIZE

```
#define ATCA_ECC204_CONFIG_SLOT_SIZE (16)
```

size of ECC204 configuration slot size

10.75.2.44 ATCA_ECC_CONFIG_SIZE

```
#define ATCA_ECC_CONFIG_SIZE (128)
```

size of configuration zone

10.75.2.45 ATCA_ECDH

```
#define ATCA_ECDH ((uint8_t)0x43)
```

ECDH command op-code.

10.75.2.46 ATCA_GENDIG

```
#define ATCA_GENDIG ((uint8_t)0x15)
```

GenDig command op-code.

10.75.2.47 ATCA_GENKEY

```
#define ATCA_GENKEY ((uint8_t)0x40)
```

GenKey command op-code.

10.75.2.48 ATCA_HMAC

```
#define ATCA_HMAC ((uint8_t)0x11)
```

HMAC command op-code.

10.75.2.49 ATCA_INFO

```
#define ATCA_INFO ((uint8_t)0x30)
```

Info command op-code.

10.75.2.50 ATCA_K283_KEY_TYPE

```
#define ATCA_K283_KEY_TYPE 1
```

K283 NIST ECC key.

10.75.2.51 ATCA_KDF

```
#define ATCA_KDF ((uint8_t)0x56)
```

KDF command op-code.

10.75.2.52 ATCA_KEY_COUNT

```
#define ATCA_KEY_COUNT (16)
```

number of keys

10.75.2.53 ATCA_KEY_ID_MAX

```
#define ATCA_KEY_ID_MAX ((uint8_t)15)
```

maximum value for key id

10.75.2.54 ATCA_KEY_SIZE

```
#define ATCA_KEY_SIZE (32)
```

size of a symmetric SHA key

10.75.2.55 ATCA_LOCK

```
#define ATCA_LOCK ((uint8_t)0x17)
```

Lock command op-code.

10.75.2.56 ATCA_LOCKED

```
#define ATCA_LOCKED (0x00)
```

Value indicating a locked zone.

10.75.2.57 ATCA_MAC

```
#define ATCA_MAC ((uint8_t)0x08)
```

MAC command op-code.

10.75.2.58 ATCA_NONCE

```
#define ATCA_NONCE ((uint8_t)0x16)
```

Nonce command op-code.

10.75.2.59 ATCA_OPCODE_IDX

```
#define ATCA_OPCODE_IDX (1)
```

command packet index for op-code

10.75.2.60 ATCA_OTP_BLOCK_MAX

```
#define ATCA_OTP_BLOCK_MAX ((uint8_t)1)
```

maximum value for OTP block

10.75.2.61 ATCA_OTP_SIZE

```
#define ATCA_OTP_SIZE (64)
```

size of OTP zone

10.75.2.62 ATCA_P256_KEY_TYPE

```
#define ATCA_P256_KEY_TYPE 4
```

P256 NIST ECC key.

10.75.2.63 ATCA_PACKET_OVERHEAD

```
#define ATCA_PACKET_OVERHEAD (ATCA_COUNT_SIZE + ATCA_CRC_SIZE)
```

Number of bytes in the command packet.

10.75.2.64 ATCA_PARAM1_IDX

```
#define ATCA_PARAM1_IDX (2)
```

command packet index for first parameter

10.75.2.65 ATCA_PARAM2_IDX

```
#define ATCA_PARAM2_IDX (3)
```

command packet index for second parameter

10.75.2.66 ATCA_PAUSE

```
#define ATCA_PAUSE ((uint8_t)0x01)
```

Pause command op-code.

10.75.2.67 ATCA_PRIV_KEY_SIZE

```
#define ATCA_PRIV_KEY_SIZE (32)
```

size of a p256 private key

10.75.2.68 ATCA_PRIVWRITE

```
#define ATCA_PRIVWRITE ((uint8_t)0x46)
```

PrivWrite command op-code.

10.75.2.69 ATCA_PUB_KEY_PAD

```
#define ATCA_PUB_KEY_PAD (4)
```

size of the public key pad

10.75.2.70 ATCA_PUB_KEY_SIZE

```
#define ATCA_PUB_KEY_SIZE (64)
```

size of a p256 public key

10.75.2.71 ATCA_RANDOM

```
#define ATCA_RANDOM ((uint8_t)0x1B)
```

Random command op-code.

10.75.2.72 ATCA_READ

```
#define ATCA_READ ((uint8_t)0x02)
```

Read command op-code.

10.75.2.73 ATCA_RSP_DATA_IDX

```
#define ATCA_RSP_DATA_IDX (1)
```

buffer index of data in response

10.75.2.74 ATCA_RSP_SIZE_16

```
#define ATCA_RSP_SIZE_16 ((uint8_t)19)
```

size of response packet containing 16 bytes data

10.75.2.75 ATCA_RSP_SIZE_32

```
#define ATCA_RSP_SIZE_32 ((uint8_t)35)
```

size of response packet containing 32 bytes data

10.75.2.76 ATCA_RSP_SIZE_4

```
#define ATCA_RSP_SIZE_4 ((uint8_t)7)
```

size of response packet containing 4 bytes data

10.75.2.77 ATCA_RSP_SIZE_64

```
#define ATCA_RSP_SIZE_64 ((uint8_t)67)
```

size of response packet containing 64 bytes data

10.75.2.78 ATCA_RSP_SIZE_72

```
#define ATCA_RSP_SIZE_72 ((uint8_t)75)
```

size of response packet containing 64 bytes data

10.75.2.79 ATCA_RSP_SIZE_MAX

```
#define ATCA_RSP_SIZE_MAX ((uint8_t)75)
```

maximum size of response packet (GenKey and Verify command)

10.75.2.80 ATCA_RSP_SIZE_MIN

```
#define ATCA_RSP_SIZE_MIN ((uint8_t)4)
```

minimum number of bytes in response

10.75.2.81 ATCA_RSP_SIZE_VAL

```
#define ATCA_RSP_SIZE_VAL ((uint8_t)7)
```

size of response packet containing four bytes of data

10.75.2.82 ATCA_SECUREBOOT

```
#define ATCA_SECUREBOOT ((uint8_t)0x80)
```

Secure Boot command op-code.

10.75.2.83 ATCA_SELFTEST

```
#define ATCA_SELFTEST ((uint8_t)0x77)
```

Self test command op-code.

10.75.2.84 ATCA_SERIAL_NUM_SIZE

```
#define ATCA_SERIAL_NUM_SIZE (9)
```

number of bytes in the device serial number

10.75.2.85 ATCA_SHA

```
#define ATCA_SHA ((uint8_t)0x47)
```

SHA command op-code.

10.75.2.86 ATCA_SHA_CONFIG_SIZE

```
#define ATCA_SHA_CONFIG_SIZE (88)
```

size of configuration zone

10.75.2.87 ATCA_SHA_DIGEST_SIZE

```
#define ATCA_SHA_DIGEST_SIZE (32)
```

10.75.2.88 ATCA_SHA_KEY_TYPE

```
#define ATCA_SHA_KEY_TYPE 7
```

SHA key or other data.

10.75.2.89 ATCA_SIG_SIZE

```
#define ATCA_SIG_SIZE (64)
```

size of a p256 signature

10.75.2.90 ATCA_SIGN

```
#define ATCA_SIGN ((uint8_t)0x41)
```

Sign command op-code.

10.75.2.91 ATCA_TEMPKEY_KEYID

```
#define ATCA_TEMPKEY_KEYID (0xFFFF)
```

KeyID when referencing TempKey.

10.75.2.92 ATCA_UNLOCKED

```
#define ATCA_UNLOCKED (0x55)
```

Value indicating an unlocked zone.

10.75.2.93 ATCA_UPDATE_EXTRA

```
#define ATCA_UPDATE_EXTRA ((uint8_t)0x20)
```

UpdateExtra command op-code.

10.75.2.94 ATCA_VERIFY

```
#define ATCA_VERIFY ((uint8_t)0x45)
```

GenKey command op-code.

10.75.2.95 ATCA_WORD_SIZE

```
#define ATCA_WORD_SIZE (4)
```

size of a word

10.75.2.96 ATCA_WRITE

```
#define ATCA_WRITE ((uint8_t)0x12)
```

Write command op-code.

10.75.2.97 ATCA_ZONE_ENCRYPTED

```
#define ATCA_ZONE_ENCRYPTED ((uint8_t)0x40)
```

Zone bit 6 set: Write is encrypted with an unlocked data zone.

10.75.2.98 ATCA_ZONE_MASK

```
#define ATCA_ZONE_MASK ((uint8_t)0x03)
```

Zone mask.

10.75.2.99 ATCA_ZONE_READWRITE_32

```
#define ATCA_ZONE_READWRITE_32 ((uint8_t)0x80)
```

Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.

10.75.2.100 CHECKMAC_CLIENT_CHALLENGE_IDX

```
#define CHECKMAC_CLIENT_CHALLENGE_IDX ATCA_DATA_IDX
```

CheckMAC command index for client challenge.

10.75.2.101 CHECKMAC_CLIENT_CHALLENGE_SIZE

```
#define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)
```

CheckMAC size of client challenge.

10.75.2.102 CHECKMAC_CLIENT_COMMAND_SIZE

```
#define CHECKMAC_CLIENT_COMMAND_SIZE (4)
```

CheckMAC size of client command header size inside "other data".

10.75.2.103 CHECKMAC_CLIENT_RESPONSE_IDX

```
#define CHECKMAC_CLIENT_RESPONSE_IDX (37)
```

CheckMAC command index for client response.

10.75.2.104 CHECKMAC_CLIENT_RESPONSE_SIZE

```
#define CHECKMAC_CLIENT_RESPONSE_SIZE (32)
```

CheckMAC size of client response.

10.75.2.105 CHECKMAC_CMD_MATCH

```
#define CHECKMAC_CMD_MATCH (0)
```

CheckMAC return value when there is a match.

10.75.2.106 CHECKMAC_CMD_MISMATCH

```
#define CHECKMAC_CMD_MISMATCH (1)
```

CheckMAC return value when there is a mismatch.

10.75.2.107 CHECKMAC_COUNT

```
#define CHECKMAC_COUNT (84)
```

CheckMAC command packet size.

10.75.2.108 CHECKMAC_DATA_IDX

```
#define CHECKMAC_DATA_IDX (69)
```

CheckMAC command index for other data.

10.75.2.109 CHECKMAC_KEYID_IDX

```
#define CHECKMAC_KEYID_IDX ATCA_PARAM2_IDX
```

CheckMAC command index for key identifier.

10.75.2.110 CHECKMAC_MODE_BLOCK1_TEMPKEY

```
#define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
```

CheckMAC mode bit 1: first SHA block from TempKey.

10.75.2.111 CHECKMAC_MODE_BLOCK2_TEMPKEY

```
#define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
```

CheckMAC mode bit 0: second SHA block from TempKey.

10.75.2.112 CHECKMAC_MODE_CHALLENGE

```
#define CHECKMAC_MODE_CHALLENGE ((uint8_t)0x00)
```

CheckMAC mode 0: first SHA block from key id.

10.75.2.113 CHECKMAC_MODE_IDX

```
#define CHECKMAC_MODE_IDX ATCA_PARAM1_IDX
```

CheckMAC command index for mode.

10.75.2.114 CHECKMAC_MODE_INCLUDE_OTP_64

```
#define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
```

CheckMAC mode bit 5: include first 64 OTP bits.

10.75.2.115 CHECKMAC_MODE_MASK

```
#define CHECKMAC_MODE_MASK ((uint8_t)0x27)
```

CheckMAC mode bits 3, 4, 6, and 7 are 0.

10.75.2.116 CHECKMAC_MODE_SOURCE_FLAG_MATCH

```
#define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
```

CheckMAC mode bit 2: match TempKey.SourceFlag.

10.75.2.117 CHECKMAC_OTHER_DATA_SIZE

```
#define CHECKMAC_OTHER_DATA_SIZE (13)
```

CheckMAC size of "other data".

10.75.2.118 CHECKMAC_RSP_SIZE

```
#define CHECKMAC_RSP_SIZE ATCA_RSP_SIZE_MIN
```

CheckMAC response packet size.

10.75.2.119 CMD_STATUS_BYTE_COMM

```
#define CMD_STATUS_BYTE_COMM ((uint8_t)0xFF)
```

communication error

10.75.2.120 CMD_STATUS_BYTE_ECC

```
#define CMD_STATUS_BYTE_ECC ((uint8_t)0x05)
```

command ECC error

10.75.2.121 CMD_STATUS_BYTE_EXEC

```
#define CMD_STATUS_BYTE_EXEC ((uint8_t)0x0F)
```

command execution error

10.75.2.122 CMD_STATUS_BYTE_PARSE

```
#define CMD_STATUS_BYTE_PARSE ((uint8_t)0x03)
```

command parse error

10.75.2.123 CMD_STATUS_SUCCESS

```
#define CMD_STATUS_SUCCESS ((uint8_t)0x00)
```

status byte for success

10.75.2.124 CMD_STATUS_WAKEUP

```
#define CMD_STATUS_WAKEUP ((uint8_t)0x11)
```

status byte after wake-up

10.75.2.125 COUNTER_COUNT

```
#define COUNTER_COUNT ATCA_CMD_SIZE_MIN
```

10.75.2.126 COUNTER_KEYID_IDX

```
#define COUNTER_KEYID_IDX ATCA_PARAM2_IDX
```

Counter command index for key id.

10.75.2.127 COUNTER_MAX_VALUE

```
#define COUNTER_MAX_VALUE ((uint32_t)2097151)
```

Counter maximum value of the counter.

10.75.2.128 COUNTER_MODE_IDX

```
#define COUNTER_MODE_IDX ATCA_PARAM1_IDX
```

Counter command index for mode.

10.75.2.129 COUNTER_MODE_INCREMENT

```
#define COUNTER_MODE_INCREMENT ((uint8_t)0x01)
```

Counter command mode for incrementing.

10.75.2.130 COUNTER_MODE_MASK

```
#define COUNTER_MODE_MASK ((uint8_t)0x01)
```

Counter mode bits 1 to 7 are 0.

10.75.2.131 COUNTER_MODE_READ

```
#define COUNTER_MODE_READ ((uint8_t)0x00)
```

Counter command mode for reading.

10.75.2.132 COUNTER_RSP_SIZE

```
#define COUNTER_RSP_SIZE ATCA_RSP_SIZE_4
```

Counter command response packet size.

10.75.2.133 COUNTER_SIZE

```
#define COUNTER_SIZE ATCA_RSP_SIZE_MIN
```

Counter size in binary.

10.75.2.134 DERIVE_KEY_COUNT_LARGE

```
#define DERIVE_KEY_COUNT_LARGE (39)
```

DeriveKey command packet size with MAC.

10.75.2.135 DERIVE_KEY_COUNT_SMALL

```
#define DERIVE_KEY_COUNT_SMALL ATCA_CMD_SIZE_MIN
```

DeriveKey command packet size without MAC.

10.75.2.136 DERIVE_KEY_MAC_IDX

```
#define DERIVE_KEY_MAC_IDX ATCA_DATA_IDX
```

DeriveKey command index for optional MAC.

10.75.2.137 DERIVE_KEY_MAC_SIZE

```
#define DERIVE_KEY_MAC_SIZE (32)
```

DeriveKey MAC size.

10.75.2.138 DERIVE_KEY_MODE

```
#define DERIVE_KEY_MODE ((uint8_t)0x04)
```

DeriveKey command mode set to 4 as in datasheet.

10.75.2.139 DERIVE_KEY_RANDOM_FLAG

```
#define DERIVE_KEY_RANDOM_FLAG ((uint8_t)4)
```

DeriveKey 1. parameter; has to match TempKey.SourceFlag.

10.75.2.140 DERIVE_KEY_RANDOM_IDX

```
#define DERIVE_KEY_RANDOM_IDX ATCA_PARAM1_IDX
```

DeriveKey command index for random bit.

10.75.2.141 DERIVE_KEY_RSP_SIZE

```
#define DERIVE_KEY_RSP_SIZE ATCA_RSP_SIZE_MIN
```

DeriveKey response packet size.

10.75.2.142 DERIVE_KEY_TARGETKEY_IDX

```
#define DERIVE_KEY_TARGETKEY_IDX ATCA_PARAM2_IDX
```

DeriveKey command index for target slot.

10.75.2.143 ECDH_COUNT

```
#define ECDH_COUNT (ATCA_CMD_SIZE_MIN + ATCA_PUB_KEY_SIZE)
```

10.75.2.144 ECDH_KEY_SIZE

```
#define ECDH_KEY_SIZE ATCA_BLOCK_SIZE
```

ECDH output data size.

10.75.2.145 ECDH_MODE_COPY_COMPATIBLE

```
#define ECDH_MODE_COPY_COMPATIBLE ((uint8_t)0x00)
```

10.75.2.146 ECDH_MODE_COPY_EEPROM_SLOT

```
#define ECDH_MODE_COPY_EEPROM_SLOT ((uint8_t)0x04)
```

10.75.2.147 ECDH_MODE_COPY_MASK

```
#define ECDH_MODE_COPY_MASK ((uint8_t)0x0C)
```

10.75.2.148 ECDH_MODE_COPY_OUTPUT_BUFFER

```
#define ECDH_MODE_COPY_OUTPUT_BUFFER ((uint8_t)0x0C)
```

10.75.2.149 ECDH_MODE_COPY_TEMP_KEY

```
#define ECDH_MODE_COPY_TEMP_KEY ((uint8_t)0x08)
```

10.75.2.150 ECDH_MODE_OUTPUT_CLEAR

```
#define ECDH_MODE_OUTPUT_CLEAR ((uint8_t)0x00)
```

10.75.2.151 ECDH_MODE_OUTPUT_ENC

```
#define ECDH_MODE_OUTPUT_ENC ((uint8_t)0x02)
```

10.75.2.152 ECDH_MODE_OUTPUT_MASK

```
#define ECDH_MODE_OUTPUT_MASK ((uint8_t)0x02)
```

10.75.2.153 ECDH_MODE_SOURCE_EEPROM_SLOT

```
#define ECDH_MODE_SOURCE_EEPROM_SLOT ((uint8_t)0x00)
```

10.75.2.154 ECDH_MODE_SOURCE_MASK

```
#define ECDH_MODE_SOURCE_MASK ((uint8_t)0x01)
```

10.75.2.155 ECDH_MODE_SOURCE_TEMPKEY

```
#define ECDH_MODE_SOURCE_TEMPKEY ((uint8_t)0x01)
```

10.75.2.156 ECDH_PREFIX_MODE

```
#define ECDH_PREFIX_MODE ((uint8_t)0x00)
```

10.75.2.157 ECDH_RSP_SIZE

```
#define ECDH_RSP_SIZE ATCA_RSP_SIZE_64
```

ECDH command packet size.

10.75.2.158 GENDIG_COUNT

```
#define GENDIG_COUNT ATCA_CMD_SIZE_MIN
```

GenDig command packet size without "other data".

10.75.2.159 GENDIG_DATA_IDX

```
#define GENDIG_DATA_IDX ATCA_DATA_IDX
```

GenDig command index for optional data.

10.75.2.160 GENDIG_KEYID_IDX

```
#define GENDIG_KEYID_IDX ATCA_PARAM2_IDX
```

GenDig command index for key id.

10.75.2.161 GENDIG_RSP_SIZE

```
#define GENDIG_RSP_SIZE ATCA_RSP_SIZE_MIN
```

GenDig command response packet size.

10.75.2.162 GENDIG_ZONE_CONFIG

```
#define GENDIG_ZONE_CONFIG ((uint8_t)0)
```

GenDig zone id config. Use KeyID to specify any of the four 256-bit blocks of the Configuration zone.

10.75.2.163 GENDIG_ZONE_COUNTER

```
#define GENDIG_ZONE_COUNTER ((uint8_t)4)
```

GenDig zone id counter. KeyID specifies the monotonic counter ID to be included in the message generation.

10.75.2.164 GENDIG_ZONE_DATA

```
#define GENDIG_ZONE_DATA ((uint8_t)2)
```

GenDig zone id data. Use KeyID to specify a slot in the Data zone or a transport key in the hardware array.

10.75.2.165 GENDIG_ZONE_IDX

```
#define GENDIG_ZONE_IDX ATCA_PARAM1_IDX
```

GenDig command index for zone.

10.75.2.166 GENDIG_ZONE_KEY_CONFIG

```
#define GENDIG_ZONE_KEY_CONFIG ((uint8_t)5)
```

GenDig zone id key config. KeyID specifies the slot for which the configuration information is to be included in the message generation.

10.75.2.167 GENDIG_ZONE_OTP

```
#define GENDIG_ZONE_OTP ((uint8_t)1)
```

GenDig zone id OTP. Use KeyID to specify either the first or second 256-bit block of the OTP zone.

10.75.2.168 GENDIG_ZONE_SHARED_NONCE

```
#define GENDIG_ZONE_SHARED_NONCE ((uint8_t)3)
```

GenDig zone id shared nonce. KeyID specifies the location of the input value in the message generation.

10.75.2.169 GENKEY_COUNT

```
#define GENKEY_COUNT ATCA_CMD_SIZE_MIN
```

GenKey command packet size without "other data".

10.75.2.170 GENKEY_COUNT_DATA

```
#define GENKEY_COUNT_DATA (10)
```

GenKey command packet size with "other data".

10.75.2.171 GENKEY_DATA_IDX

```
#define GENKEY_DATA_IDX (5)
```

GenKey command index for other data.

10.75.2.172 GENKEY_KEYID_IDX

```
#define GENKEY_KEYID_IDX ATCA_PARAM2_IDX
```

GenKey command index for key id.

10.75.2.173 GENKEY_MODE_DIGEST

```
#define GENKEY_MODE_DIGEST ((uint8_t)0x08)
```

GenKey mode: PubKey digest will be created after the public key is calculated.

10.75.2.174 GENKEY_MODE_IDX

```
#define GENKEY_MODE_IDX ATCA_PARAM1_IDX
```

GenKey command index for mode.

10.75.2.175 GENKEY_MODE_MAC

```
#define GENKEY_MODE_MAC ((uint8_t)0x20)
```

Genkey mode: Calculate MAC of public key + session key.

10.75.2.176 GENKEY_MODE_MASK

```
#define GENKEY_MODE_MASK ((uint8_t)0x1C)
```

GenKey mode bits 0 to 1 and 5 to 7 are 0.

10.75.2.177 GENKEY_MODE_PRIVATE

```
#define GENKEY_MODE_PRIVATE ((uint8_t)0x04)
```

GenKey mode: private key generation.

10.75.2.178 GENKEY_MODE_PUBKEY_DIGEST

```
#define GENKEY_MODE_PUBKEY_DIGEST ((uint8_t)0x10)
```

GenKey mode: Calculate PubKey digest on the public key in KeyId.

10.75.2.179 GENKEY_MODE_PUBLIC

```
#define GENKEY_MODE_PUBLIC ((uint8_t)0x00)
```

GenKey mode: public key calculation.

10.75.2.180 GENKEY_OTHER_DATA_SIZE

```
#define GENKEY_OTHER_DATA_SIZE (3)
```

GenKey size of "other data".

10.75.2.181 GENKEY_PRIVATE_TO_TEMPKEY

```
#define GENKEY_PRIVATE_TO_TEMPKEY ((uint16_t)0xFFFF)
```

GenKey Create private key and store to tempkey (608 only)

10.75.2.182 GENKEY_RSP_SIZE_LONG

```
#define GENKEY_RSP_SIZE_LONG ATCA_RSP_SIZE_64
```

GenKey response packet size when returning a public key.

10.75.2.183 GENKEY_RSP_SIZE_SHORT

```
#define GENKEY_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

GenKey response packet size in Digest mode.

10.75.2.184 HMAC_COUNT

```
#define HMAC_COUNT ATCA_CMD_SIZE_MIN
```

HMAC command packet size.

10.75.2.185 HMAC_DIGEST_SIZE

```
#define HMAC_DIGEST_SIZE (32)
```

HMAC size of digest response.

10.75.2.186 HMAC_KEYID_IDX

```
#define HMAC_KEYID_IDX ATCA_PARAM2_IDX
```

HMAC command index for key id.

10.75.2.187 HMAC_MODE_FLAG_FULLSN

```
#define HMAC_MODE_FLAG_FULLSN ((uint8_t)0x40)
```

HMAC mode bit 6: If set, include the 48 bits SN[2:3] and SN[4:7] in the message.; otherwise, the corresponding message bits are set to zero.

10.75.2.188 HMAC_MODE_FLAG_OTP64

```
#define HMAC_MODE_FLAG_OTP64 ((uint8_t)0x20)
```

HMAC mode bit 5: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message.; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored. Not applicable for ATECC508A.

10.75.2.189 HMAC_MODE_FLAG_OTP88

```
#define HMAC_MODE_FLAG_OTP88 ((uint8_t)0x10)
```

HMAC mode bit 4: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message.; otherwise, the corresponding message bits are set to zero. Not applicable for ATECC508A.

10.75.2.190 HMAC_MODE_FLAG_TK_NORAND

```
#define HMAC_MODE_FLAG_TK_NORAND ((uint8_t)0x04)
```

HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.

10.75.2.191 HMAC_MODE_FLAG_TK_RAND

```
#define HMAC_MODE_FLAG_TK_RAND ((uint8_t)0x00)
```

HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.

10.75.2.192 HMAC_MODE_IDX

```
#define HMAC_MODE_IDX ATCA_PARAM1_IDX
```

HMAC command index for mode.

10.75.2.193 HMAC_MODE_MASK

```
#define HMAC_MODE_MASK ((uint8_t)0x74)
```

HMAC mode bits 0, 1, 3, and 7 are 0.

10.75.2.194 HMAC_RSP_SIZE

```
#define HMAC_RSP_SIZE ATCA_RSP_SIZE_32
```

HMAC command response packet size.

10.75.2.195 INFO_COUNT

```
#define INFO_COUNT ATCA_CMD_SIZE_MIN
```

Info command packet size.

10.75.2.196 INFO_DRIVER_STATE_MASK

```
#define INFO_DRIVER_STATE_MASK ((uint8_t)0x02)
```

Info driver state mask.

10.75.2.197 INFO_MODE_GPIO

```
#define INFO_MODE_GPIO ((uint8_t)0x03)
```

Info mode GPIO.

10.75.2.198 INFO_MODE_KEY_VALID

```
#define INFO_MODE_KEY_VALID ((uint8_t)0x01)
```

Info mode KeyValid.

10.75.2.199 INFO_MODE_LOCK_STATUS

```
#define INFO_MODE_LOCK_STATUS ((uint8_t)0x02)
```

Info mode Lock status for ECC204 device.

10.75.2.200 INFO_MODE_MAX

```
#define INFO_MODE_MAX ((uint8_t)0x03)
```

Info mode maximum value.

10.75.2.201 INFO_MODE_REVISION

```
#define INFO_MODE_REVISION ((uint8_t)0x00)
```

Info mode Revision.

10.75.2.202 INFO_MODE_STATE

```
#define INFO_MODE_STATE ((uint8_t)0x02)
```

Info mode State.

10.75.2.203 INFO_MODE_VOL_KEY_PERMIT

```
#define INFO_MODE_VOL_KEY_PERMIT ((uint8_t)0x04)
```

Info mode GPIO.

10.75.2.204 INFO_NO_STATE

```
#define INFO_NO_STATE ((uint8_t)0x00)
```

Info mode is not the state mode.

10.75.2.205 INFO_OUTPUT_STATE_MASK

```
#define INFO_OUTPUT_STATE_MASK ((uint8_t)0x01)
```

Info output state mask.

10.75.2.206 INFO_PARAM1_IDX

```
#define INFO_PARAM1_IDX ATCA_PARAM1_IDX
```

Info command index for 1. parameter.

10.75.2.207 INFO_PARAM2_IDX

```
#define INFO_PARAM2_IDX ATCA_PARAM2_IDX
```

Info command index for 2. parameter.

10.75.2.208 INFO_PARAM2_LATCH_CLEAR

```
#define INFO_PARAM2_LATCH_CLEAR ((uint16_t)0x0000)
```

Info param2 to clear the persistent latch.

10.75.2.209 INFO_PARAM2_LATCH_SET

```
#define INFO_PARAM2_LATCH_SET ((uint16_t)0x0001)
```

Info param2 to set the persistent latch.

10.75.2.210 INFO_PARAM2_SET_LATCH_STATE

```
#define INFO_PARAM2_SET_LATCH_STATE ((uint16_t)0x0002)
```

Info param2 to set the persistent latch state.

10.75.2.211 INFO_RSP_SIZE

```
#define INFO_RSP_SIZE ATCA_RSP_SIZE_VAL
```

Info command response packet size.

10.75.2.212 INFO_SIZE

```
#define INFO_SIZE ((uint8_t)0x04)
```

Info return size.

10.75.2.213 KDF_DETAILS_AES_KEY_LOC_MASK

```
#define KDF_DETAILS_AES_KEY_LOC_MASK ((uint32_t)0x00000003)
```

KDF details for AES, key location mask.

10.75.2.214 KDF_DETAILS_HKDF_MSG_LOC_INPUT

```
#define KDF_DETAILS_HKDF_MSG_LOC_INPUT ((uint32_t)0x00000002)
```

KDF details for HKDF, message location in input parameter.

10.75.2.215 KDF_DETAILS_HKDF_MSG_LOC_IV

```
#define KDF_DETAILS_HKDF_MSG_LOC_IV ((uint32_t)0x00000003)
```

KDF details for HKDF, message location is a special IV function.

10.75.2.216 KDF_DETAILS_HKDF_MSG_LOC_MASK

```
#define KDF_DETAILS_HKDF_MSG_LOC_MASK ((uint32_t)0x00000003)
```

KDF details for HKDF, message location mask.

10.75.2.217 KDF_DETAILS_HKDF_MSG_LOC_SLOT

```
#define KDF_DETAILS_HKDF_MSG_LOC_SLOT ((uint32_t)0x00000000)
```

KDF details for HKDF, message location in slot.

10.75.2.218 KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY

```
#define KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY ((uint32_t)0x00000001)
```

KDF details for HKDF, message location in TempKey.

10.75.2.219 KDF_DETAILS_HKDF_ZERO_KEY

```
#define KDF_DETAILS_HKDF_ZERO_KEY ((uint32_t)0x00000004)
```

KDF details for HKDF, key is 32 bytes of zero.

10.75.2.220 KDF_DETAILS_IDX

```
#define KDF_DETAILS_IDX ATCA_DATA_IDX
```

KDF command index for details.

10.75.2.221 KDF_DETAILS_PRF_AEAD_MASK

```
#define KDF_DETAILS_PRF_AEAD_MASK ((uint32_t)0x00000600)
```

KDF details for PRF, AEAD processing mask.

10.75.2.222 KDF_DETAILS_PRF_AEAD_MODE0

```
#define KDF_DETAILS_PRF_AEAD_MODE0 ((uint32_t)0x00000000)
```

KDF details for PRF, AEAD no processing.

10.75.2.223 KDF_DETAILS_PRF_AEAD_MODE1

```
#define KDF_DETAILS_PRF_AEAD_MODE1 ((uint32_t)0x00000200)
```

KDF details for PRF, AEAD First 32 go to target, second 32 go to output buffer.

10.75.2.224 KDF_DETAILS_PRF_KEY_LEN_16

```
#define KDF_DETAILS_PRF_KEY_LEN_16 ((uint32_t)0x00000000)
```

KDF details for PRF, source key length is 16 bytes.

10.75.2.225 KDF_DETAILS_PRF_KEY_LEN_32

```
#define KDF_DETAILS_PRF_KEY_LEN_32 ((uint32_t)0x00000001)
```

KDF details for PRF, source key length is 32 bytes.

10.75.2.226 KDF_DETAILS_PRF_KEY_LEN_48

```
#define KDF_DETAILS_PRF_KEY_LEN_48 ((uint32_t)0x00000002)
```

KDF details for PRF, source key length is 48 bytes.

10.75.2.227 KDF_DETAILS_PRF_KEY_LEN_64

```
#define KDF_DETAILS_PRF_KEY_LEN_64 ((uint32_t)0x00000003)
```

KDF details for PRF, source key length is 64 bytes.

10.75.2.228 KDF_DETAILS_PRF_KEY_LEN_MASK

```
#define KDF_DETAILS_PRF_KEY_LEN_MASK ((uint32_t)0x00000003)
```

KDF details for PRF, source key length mask.

10.75.2.229 KDF_DETAILS_PRF_TARGET_LEN_32

```
#define KDF_DETAILS_PRF_TARGET_LEN_32 ((uint32_t)0x00000000)
```

KDF details for PRF, target length is 32 bytes.

10.75.2.230 KDF_DETAILS_PRF_TARGET_LEN_64

```
#define KDF_DETAILS_PRF_TARGET_LEN_64 ((uint32_t)0x00000100)
```

KDF details for PRF, target length is 64 bytes.

10.75.2.231 KDF_DETAILS_PRF_TARGET_LEN_MASK

```
#define KDF_DETAILS_PRF_TARGET_LEN_MASK ((uint32_t)0x00000100)
```

KDF details for PRF, target length mask.

10.75.2.232 KDF_DETAILS_SIZE

```
#define KDF_DETAILS_SIZE 4
```

KDF details (param3) size.

10.75.2.233 KDF_KEYID_IDX

```
#define KDF_KEYID_IDX ATCA_PARAM2_IDX
```

KDF command index for key id.

10.75.2.234 KDF_MESSAGE_IDX

```
#define KDF_MESSAGE_IDX (ATCA_DATA_IDX + KDF_DETAILS_SIZE)
```

10.75.2.235 KDF_MODE_ALG_AES

```
#define KDF_MODE_ALG_AES ((uint8_t)0x20)
```

KDF mode AES algorithm.

10.75.2.236 KDF_MODE_ALG_HKDF

```
#define KDF_MODE_ALG_HKDF ((uint8_t)0x40)
```

KDF mode HKDF algorithm.

10.75.2.237 KDF_MODE_ALG_MASK

```
#define KDF_MODE_ALG_MASK ((uint8_t)0x60)
```

KDF mode algorithm mask.

10.75.2.238 KDF_MODE_ALG_PRF

```
#define KDF_MODE_ALG_PRF ((uint8_t)0x00)
```

KDF mode PRF algorithm.

10.75.2.239 KDF_MODE_IDX

```
#define KDF_MODE_IDX ATCA_PARAM1_IDX
```

KDF command index for mode.

10.75.2.240 KDF_MODE_SOURCE_ALTKEYBUF

```
#define KDF_MODE_SOURCE_ALTKEYBUF ((uint8_t)0x03)
```

KDF mode source key in alternate key buffer.

10.75.2.241 KDF_MODE_SOURCE_MASK

```
#define KDF_MODE_SOURCE_MASK ((uint8_t)0x03)
```

KDF mode source key mask.

10.75.2.242 KDF_MODE_SOURCE_SLOT

```
#define KDF_MODE_SOURCE_SLOT ((uint8_t)0x02)
```

KDF mode source key in a slot.

10.75.2.243 KDF_MODE_SOURCE_TEMPKEY

```
#define KDF_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

KDF mode source key in TempKey.

10.75.2.244 KDF_MODE_SOURCE_TEMPKEY_UP

```
#define KDF_MODE_SOURCE_TEMPKEY_UP ((uint8_t)0x01)
```

KDF mode source key in upper TempKey.

10.75.2.245 KDF_MODE_TARGET_ALTKEYBUF

```
#define KDF_MODE_TARGET_ALTKEYBUF ((uint8_t)0x0C)
```

KDF mode target key in alternate key buffer.

10.75.2.246 KDF_MODE_TARGET_MASK

```
#define KDF_MODE_TARGET_MASK ((uint8_t)0x1C)
```

KDF mode target key mask.

10.75.2.247 KDF_MODE_TARGET_OUTPUT

```
#define KDF_MODE_TARGET_OUTPUT ((uint8_t)0x10)
```

KDF mode target key in output buffer.

10.75.2.248 KDF_MODE_TARGET_OUTPUT_ENC

```
#define KDF_MODE_TARGET_OUTPUT_ENC ((uint8_t)0x14)
```

KDF mode target key encrypted in output buffer.

10.75.2.249 KDF_MODE_TARGET_SLOT

```
#define KDF_MODE_TARGET_SLOT ((uint8_t)0x08)
```

KDF mode target key in slot.

10.75.2.250 KDF_MODE_TARGET_TEMPKEY

```
#define KDF_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

KDF mode target key in TempKey.

10.75.2.251 KDF_MODE_TARGET_TEMPKEY_UP

```
#define KDF_MODE_TARGET_TEMPKEY_UP ((uint8_t)0x04)
```

KDF mode target key in upper TempKey.

10.75.2.252 LOCK_COUNT

```
#define LOCK_COUNT ATCA_CMD_SIZE_MIN
```

Lock command packet size.

10.75.2.253 LOCK_ECC204_ZONE_CONFIG

```
#define LOCK_ECC204_ZONE_CONFIG ((uint8_t)0x01)
```

Lock ECC204 configuration zone by slot.

10.75.2.254 LOCK_ECC204_ZONE_DATA

```
#define LOCK_ECC204_ZONE_DATA ((uint8_t)0x00)
```

Lock ECC204 Data zone by slot.

10.75.2.255 LOCK_RSP_SIZE

```
#define LOCK_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Lock command response packet size.

10.75.2.256 LOCK_SUMMARY_IDX

```
#define LOCK_SUMMARY_IDX ATCA_PARAM2_IDX
```

Lock command index for summary.

10.75.2.257 LOCK_ZONE_CONFIG

```
#define LOCK_ZONE_CONFIG ((uint8_t)0x00)
```

Lock zone is Config.

10.75.2.258 LOCK_ZONE_DATA

```
#define LOCK_ZONE_DATA ((uint8_t)0x01)
```

Lock zone is OTP or Data.

10.75.2.259 LOCK_ZONE_DATA_SLOT

```
#define LOCK_ZONE_DATA_SLOT ((uint8_t)0x02)
```

Lock slot of Data.

10.75.2.260 LOCK_ZONE_IDX

```
#define LOCK_ZONE_IDX ATCA_PARAM1_IDX
```

Lock command index for zone.

10.75.2.261 LOCK_ZONE_MASK

```
#define LOCK_ZONE_MASK (0xBF)
```

Lock parameter 1 bits 6 are 0.

10.75.2.262 LOCK_ZONE_NO_CRC

```
#define LOCK_ZONE_NO_CRC ((uint8_t)0x80)
```

Lock command: Ignore summary.

10.75.2.263 MAC_CHALLENGE_IDX

```
#define MAC_CHALLENGE_IDX ATCA_DATA_IDX
```

MAC command index for optional challenge.

10.75.2.264 MAC_CHALLENGE_SIZE

```
#define MAC_CHALLENGE_SIZE (32)
```

MAC size of challenge.

10.75.2.265 MAC_COUNT_LONG

```
#define MAC_COUNT_LONG (39)
```

MAC command packet size with challenge.

10.75.2.266 MAC_COUNT_SHORT

```
#define MAC_COUNT_SHORT ATCA_CMD_SIZE_MIN
```

MAC command packet size without challenge.

10.75.2.267 MAC_KEYID_IDX

```
#define MAC_KEYID_IDX ATCA_PARAM2_IDX
```

MAC command index for key id.

10.75.2.268 MAC_MODE_BLOCK1_TEMPKEY

```
#define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
```

MAC mode bit 1: first SHA block from TempKey.

10.75.2.269 MAC_MODE_BLOCK2_TEMPKEY

```
#define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
```

MAC mode bit 0: second SHA block from TempKey.

10.75.2.270 MAC_MODE_CHALLENGE

```
#define MAC_MODE_CHALLENGE ((uint8_t)0x00)
```

MAC mode 0: first SHA block from data slot.

10.75.2.271 MAC_MODE_IDX

```
#define MAC_MODE_IDX ATCA_PARAM1_IDX
```

MAC command index for mode.

10.75.2.272 MAC_MODE_INCLUDE_OTP_64

```
#define MAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
```

MAC mode bit 5: include first 64 OTP bits.

10.75.2.273 MAC_MODE_INCLUDE_OTP_88

```
#define MAC_MODE_INCLUDE_OTP_88 ((uint8_t)0x10)
```

MAC mode bit 4: include first 88 OTP bits.

10.75.2.274 MAC_MODE_INCLUDE_SN

```
#define MAC_MODE_INCLUDE_SN ((uint8_t)0x40)
```

MAC mode bit 6: include serial number.

10.75.2.275 MAC_MODE_MASK

```
#define MAC_MODE_MASK ((uint8_t)0x77)
```

MAC mode bits 3 and 7 are 0.

10.75.2.276 MAC_MODE_PASSTHROUGH

```
#define MAC_MODE_PASSTHROUGH ((uint8_t)0x07)
```

MAC mode bit 0-2: pass-through mode.

10.75.2.277 MAC_MODE_PTNONCE_TEMPKEY

```
#define MAC_MODE_PTNONCE_TEMPKEY ((uint8_t)0x06)
```

MAC mode bit 0: second SHA block from TempKey.

10.75.2.278 MAC_MODE_SOURCE_FLAG_MATCH

```
#define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
```

MAC mode bit 2: match TempKey.SourceFlag.

10.75.2.279 MAC_RSP_SIZE

```
#define MAC_RSP_SIZE ATCA_RSP_SIZE_32
```

MAC command response packet size.

10.75.2.280 MAC_SIZE

```
#define MAC_SIZE (32)
```

MAC size of response.

10.75.2.281 NONCE_COUNT_LONG

```
#define NONCE_COUNT_LONG (ATCA_CMD_SIZE_MIN + 32)
```

Nonce command packet size for 32 bytes of NumIn.

10.75.2.282 NONCE_COUNT_LONG_64

```
#define NONCE_COUNT_LONG_64 (ATCA_CMD_SIZE_MIN + 64)
```

Nonce command packet size for 64 bytes of NumIn.

10.75.2.283 NONCE_COUNT_SHORT

```
#define NONCE_COUNT_SHORT (ATCA_CMD_SIZE_MIN + 20)
```

Nonce command packet size for 20 bytes of NumIn.

10.75.2.284 NONCE_INPUT_IDX

```
#define NONCE_INPUT_IDX ATCA_DATA_IDX
```

Nonce command index for input data.

10.75.2.285 NONCE_MODE_GEN_SESSION_KEY

```
#define NONCE_MODE_GEN_SESSION_KEY ((uint8_t)0x02)
```

NOnce mode: Generate session key in ECC204 device.

10.75.2.286 NONCE_MODE_IDX

```
#define NONCE_MODE_IDX ATCA_PARAM1_IDX
```

Nonce command index for mode.

10.75.2.287 NONCE_MODE_INPUT_LEN_32

```
#define NONCE_MODE_INPUT_LEN_32 ((uint8_t)0x00)
```

Nonce mode: input size is 32 bytes.

10.75.2.288 NONCE_MODE_INPUT_LEN_64

```
#define NONCE_MODE_INPUT_LEN_64 ((uint8_t)0x20)
```

Nonce mode: input size is 64 bytes.

10.75.2.289 NONCE_MODE_INPUT_LEN_MASK

```
#define NONCE_MODE_INPUT_LEN_MASK ((uint8_t)0x20)
```

Nonce mode: input size mask.

10.75.2.290 NONCE_MODE_INVALID

```
#define NONCE_MODE_INVALID ((uint8_t)0x02)
```

Nonce mode 2 is invalid.

10.75.2.291 NONCE_MODE_MASK

```
#define NONCE_MODE_MASK ((uint8_t)0x03)
```

Nonce mode bits 2 to 7 are 0.

10.75.2.292 NONCE_MODE_NO_SEED_UPDATE

```
#define NONCE_MODE_NO_SEED_UPDATE ((uint8_t)0x01)
```

Nonce mode: do not update seed.

10.75.2.293 NONCE_MODE_PASSTHROUGH

```
#define NONCE_MODE_PASSTHROUGH ((uint8_t)0x03)
```

Nonce mode: pass-through.

10.75.2.294 NONCE_MODE_SEED_UPDATE

```
#define NONCE_MODE_SEED_UPDATE ((uint8_t)0x00)
```

Nonce mode: update seed.

10.75.2.295 NONCE_MODE_TARGET_ALTKEYBUF

```
#define NONCE_MODE_TARGET_ALTKEYBUF ((uint8_t)0x80)
```

Nonce mode: target is Alternate Key Buffer.

10.75.2.296 NONCE_MODE_TARGET_MASK

```
#define NONCE_MODE_TARGET_MASK ((uint8_t)0xC0)
```

Nonce mode: target mask.

10.75.2.297 NONCE_MODE_TARGET_MSGDIGBUF

```
#define NONCE_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
```

Nonce mode: target is Message Digest Buffer.

10.75.2.298 NONCE_MODE_TARGET_TEMPKEY

```
#define NONCE_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

Nonce mode: target is TempKey.

10.75.2.299 NONCE_NUMIN_SIZE

```
#define NONCE_NUMIN_SIZE (20)
```

Nonce NumIn size for random modes.

10.75.2.300 NONCE_NUMIN_SIZE_PASSTHROUGH

```
#define NONCE_NUMIN_SIZE_PASSTHROUGH (32)
```

Nonce NumIn size for 32-byte pass-through mode.

10.75.2.301 NONCE_PARAM2_IDX

```
#define NONCE_PARAM2_IDX ATCA_PARAM2_IDX
```

Nonce command index for 2. parameter.

10.75.2.302 NONCE_RSP_SIZE_LONG

```
#define NONCE_RSP_SIZE_LONG ATCA_RSP_SIZE_32
```

Nonce command response packet size with output.

10.75.2.303 NONCE_RSP_SIZE_SHORT

```
#define NONCE_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

Nonce command response packet size with no output.

10.75.2.304 NONCE_ZERO_CALC_MASK

```
#define NONCE_ZERO_CALC_MASK ((uint16_t)0x8000)
```

Nonce zero (param2): calculation mode mask.

10.75.2.305 NONCE_ZERO_CALC_RANDOM

```
#define NONCE_ZERO_CALC_RANDOM ((uint16_t)0x0000)
```

Nonce zero (param2): calculation mode random, use RNG in calculation and return RNG output.

10.75.2.306 NONCE_ZERO_CALC_TEMPKEY

```
#define NONCE_ZERO_CALC_TEMPKEY ((uint16_t)0x8000)
```

Nonce zero (param2): calculation mode TempKey, use TempKey in calculation and return new TempKey value.

10.75.2.307 OUTNONCE_SIZE

```
#define OUTNONCE_SIZE (32)
```

Size of the OutNonce response expected from several commands.

10.75.2.308 PAUSE_COUNT

```
#define PAUSE_COUNT ATCA_CMD_SIZE_MIN
```

Pause command packet size.

10.75.2.309 PAUSE_PARAM2_IDX

```
#define PAUSE_PARAM2_IDX ATCA_PARAM2_IDX
```

Pause command index for 2. parameter.

10.75.2.310 PAUSE_RSP_SIZE

```
#define PAUSE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Pause command response packet size.

10.75.2.311 PAUSE_SELECT_IDX

```
#define PAUSE_SELECT_IDX ATCA_PARAM1_IDX
```

Pause command index for Selector.

10.75.2.312 PRIVWRITE_COUNT

```
#define PRIVWRITE_COUNT (75)
```

PrivWrite command packet size.

10.75.2.313 PRIVWRITE_KEYID_IDX

```
#define PRIVWRITE_KEYID_IDX ATCA_PARAM2_IDX
```

PrivWrite command index for KeyID.

10.75.2.314 PRIVWRITE_MAC_IDX

```
#define PRIVWRITE_MAC_IDX (41)
```

PrivWrite command index for MAC.

10.75.2.315 PRIVWRITE_MODE_ENCRYPT

```
#define PRIVWRITE_MODE_ENCRYPT ((uint8_t)0x40)
```

PrivWrite mode: encrypted.

10.75.2.316 PRIVWRITE_RSP_SIZE

```
#define PRIVWRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

PrivWrite command response packet size.

10.75.2.317 PRIVWRITE_VALUE_IDX

```
#define PRIVWRITE_VALUE_IDX ( 5)
```

PrivWrite command index for value.

10.75.2.318 PRIVWRITE_ZONE_IDX

```
#define PRIVWRITE_ZONE_IDX ATCA_PARAM1_IDX
```

PrivWrite command index for zone.

10.75.2.319 PRIVWRITE_ZONE_MASK

```
#define PRIVWRITE_ZONE_MASK ((uint8_t)0x40)
```

PrivWrite zone bits 0 to 5 and 7 are 0.

10.75.2.320 RANDOM_COUNT

```
#define RANDOM_COUNT ATCA_CMD_SIZE_MIN
```

Random command packet size.

10.75.2.321 RANDOM_MODE_IDX

```
#define RANDOM_MODE_IDX ATCA_PARAM1_IDX
```

Random command index for mode.

10.75.2.322 RANDOM_NO_SEED_UPDATE

```
#define RANDOM_NO_SEED_UPDATE ((uint8_t)0x01)
```

Random mode for no seed update.

10.75.2.323 RANDOM_NUM_SIZE

```
#define RANDOM_NUM_SIZE ((uint8_t)32)
```

Number of bytes in the data packet of a random command.

10.75.2.324 RANDOM_PARAM2_IDX

```
#define RANDOM_PARAM2_IDX ATCA_PARAM2_IDX
```

Random command index for 2. parameter.

10.75.2.325 RANDOM_RSP_SIZE

```
#define RANDOM_RSP_SIZE ATCA_RSP_SIZE_32
```

Random command response packet size.

10.75.2.326 RANDOM_SEED_UPDATE

```
#define RANDOM_SEED_UPDATE ((uint8_t)0x00)
```

Random mode for automatic seed update.

10.75.2.327 READ_32_RSP_SIZE

```
#define READ_32_RSP_SIZE ATCA_RSP_SIZE_32
```

Read command response packet size when reading 32 bytes.

10.75.2.328 READ_4_RSP_SIZE

```
#define READ_4_RSP_SIZE ATCA_RSP_SIZE_VAL
```

Read command response packet size when reading 4 bytes.

10.75.2.329 READ_ADDR_IDX

```
#define READ_ADDR_IDX ATCA_PARAM2_IDX
```

Read command index for address.

10.75.2.330 READ_COUNT

```
#define READ_COUNT ATCA_CMD_SIZE_MIN
```

Read command packet size.

10.75.2.331 READ_ZONE_IDX

```
#define READ_ZONE_IDX ATCA_PARAM1_IDX
```

Read command index for zone.

10.75.2.332 READ_ZONE_MASK

```
#define READ_ZONE_MASK ((uint8_t)0x83)
```

Read zone bits 2 to 6 are 0.

10.75.2.333 RSA2048_KEY_SIZE

```
#define RSA2048_KEY_SIZE (256)
```

size of a RSA private key

10.75.2.334 SECUREBOOT_COUNT_DIG

```
#define SECUREBOOT_COUNT_DIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE)
```

SecureBoot command packet size for just a digest.

10.75.2.335 SECUREBOOT_COUNT_DIG_SIG

```
#define SECUREBOOT_COUNT_DIG_SIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE + SECUREBOOT_SIGNATURE_SIZE)
```

SecureBoot command packet size for a digest and signature.

10.75.2.336 SECUREBOOT_DIGEST_SIZE

```
#define SECUREBOOT_DIGEST_SIZE (32)
```

SecureBoot digest input size.

10.75.2.337 SECUREBOOT_MAC_SIZE

```
#define SECUREBOOT_MAC_SIZE (32)
```

SecureBoot MAC output size.

10.75.2.338 SECUREBOOT_MODE_ENC_MAC_FLAG

```
#define SECUREBOOT_MODE_ENC_MAC_FLAG ((uint8_t)0x80)
```

SecureBoot mode flag for encrypted digest and returning validating MAC.

10.75.2.339 SECUREBOOT_MODE_FULL

```
#define SECUREBOOT_MODE_FULL ((uint8_t)0x05)
```

SecureBoot mode Full.

10.75.2.340 SECUREBOOT_MODE_FULL_COPY

```
#define SECUREBOOT_MODE_FULL_COPY ((uint8_t)0x07)
```

SecureBoot mode FullCopy.

10.75.2.341 SECUREBOOT_MODE_FULL_STORE

```
#define SECUREBOOT_MODE_FULL_STORE ((uint8_t)0x06)
```

SecureBoot mode FullStore.

10.75.2.342 SECUREBOOT_MODE_IDX

```
#define SECUREBOOT_MODE_IDX ATCA_PARAM1_IDX
```

SecureBoot command index for mode.

10.75.2.343 SECUREBOOT_MODE_MASK

```
#define SECUREBOOT_MODE_MASK ((uint8_t)0x07)
```

SecureBoot mode mask.

10.75.2.344 SECUREBOOT_MODE_PROHIBIT_FLAG

```
#define SECUREBOOT_MODE_PROHIBIT_FLAG ((uint8_t)0x40)
```

SecureBoot mode flag to prohibit SecureBoot until next power cycle.

10.75.2.345 SECUREBOOT_RSP_SIZE_MAC

```
#define SECUREBOOT_RSP_SIZE_MAC (ATCA_PACKET_OVERHEAD + SECUREBOOT_MAC_SIZE)
```

SecureBoot response packet size with MAC.

10.75.2.346 SECUREBOOT_RSP_SIZE_NO_MAC

```
#define SECUREBOOT_RSP_SIZE_NO_MAC ATCA_RSP_SIZE_MIN
```

SecureBoot response packet size for no MAC.

10.75.2.347 SECUREBOOT_SIGNATURE_SIZE

```
#define SECUREBOOT_SIGNATURE_SIZE (64)
```

SecureBoot signature input size.

10.75.2.348 SECUREBOOTCONFIG_MODE_DISABLED

```
#define SECUREBOOTCONFIG_MODE_DISABLED ((uint16_t)0x0000)
```

Disabled SecureBootMode in SecureBootConfig value.

10.75.2.349 SECUREBOOTCONFIG_MODE_FULL_BOTH

```
#define SECUREBOOTCONFIG_MODE_FULL_BOTH ((uint16_t)0x0001)
```

Both digest and signature always required SecureBootMode in SecureBootConfig value.

10.75.2.350 SECUREBOOTCONFIG_MODE_FULL_DIG

```
#define SECUREBOOTCONFIG_MODE_FULL_DIG ((uint16_t)0x0003)
```

Digest stored SecureBootMode in SecureBootConfig value.

10.75.2.351 SECUREBOOTCONFIG_MODE_FULL_SIG

```
#define SECUREBOOTCONFIG_MODE_FULL_SIG ((uint16_t)0x0002)
```

Signature stored SecureBootMode in SecureBootConfig value.

10.75.2.352 SECUREBOOTCONFIG_MODE_MASK

```
#define SECUREBOOTCONFIG_MODE_MASK ((uint16_t)0x0003)
```

Mask for SecureBootMode field in SecureBootConfig value.

10.75.2.353 SECUREBOOTCONFIG_OFFSET

```
#define SECUREBOOTCONFIG_OFFSET (70)
```

SecureBootConfig byte offset into the configuration zone.

10.75.2.354 SELFTEST_COUNT

```
#define SELFTEST_COUNT ATCA_CMD_SIZE_MIN
```

SelfTest command packet size.

10.75.2.355 SELFTEST_MODE_AES

```
#define SELFTEST_MODE_AES ((uint8_t)0x10)
```

SelfTest mode AES encrypt function.

10.75.2.356 SELFTEST_MODE_ALL

```
#define SELFTEST_MODE_ALL ((uint8_t)0x3B)
```

SelfTest mode all algorithms.

10.75.2.357 SELFTEST_MODE_ECDH

```
#define SELFTEST_MODE_ECDH ((uint8_t)0x08)
```

SelfTest mode ECDH function.

10.75.2.358 SELFTEST_MODE_ECDSA_SIGN_VERIFY

```
#define SELFTEST_MODE_ECDSA_SIGN_VERIFY ((uint8_t)0x02)
```

SelfTest mode ECDSA verify function.

10.75.2.359 SELFTEST_MODE_IDX

```
#define SELFTEST_MODE_IDX ATCA_PARAM1_IDX
```

SelfTest command index for mode.

10.75.2.360 SELFTEST_MODE_RNG

```
#define SELFTEST_MODE_RNG ((uint8_t)0x01)
```

SelfTest mode RNG DRBG function.

10.75.2.361 SELFTEST_MODE_SHA

```
#define SELFTEST_MODE_SHA ((uint8_t)0x20)
```

SelfTest mode SHA function.

10.75.2.362 SELFTEST_RSP_SIZE

```
#define SELFTEST_RSP_SIZE ATCA_RSP_SIZE_MIN
```

SelfTest command response packet size.

10.75.2.363 SHA_COUNT_LONG

```
#define SHA_COUNT_LONG ATCA_CMD_SIZE_MIN
```

Just a starting size.

10.75.2.364 SHA_COUNT_SHORT

```
#define SHA_COUNT_SHORT ATCA_CMD_SIZE_MIN
```

10.75.2.365 SHA_DATA_MAX

```
#define SHA_DATA_MAX (64)
```

10.75.2.366 SHA_MODE_608_HMAC_END

```
#define SHA_MODE_608_HMAC_END ((uint8_t)0x02)
```

Complete the HMAC computation and return digest... Different command on 608.

10.75.2.367 SHA_MODE_ECC204_HMAC_END

```
#define SHA_MODE_ECC204_HMAC_END ((uint8_t)0x02)
```

Complete the HMAC computation and return digest... Different mode on ECC204.

10.75.2.368 SHA_MODE_ECC204_HMAC_START

```
#define SHA_MODE_ECC204_HMAC_START ((uint8_t)0x03)
```

Initialization, HMAC calculation for ECC204.

10.75.2.369 SHA_MODE_HMAC_END

```
#define SHA_MODE_HMAC_END ((uint8_t)0x05)
```

Complete the HMAC computation and return digest.

10.75.2.370 SHA_MODE_HMAC_START

```
#define SHA_MODE_HMAC_START ((uint8_t)0x04)
```

Initialization, HMAC calculation.

10.75.2.371 SHA_MODE_HMAC_UPDATE

```
#define SHA_MODE_HMAC_UPDATE ((uint8_t)0x01)
```

Add 64 bytes in the meesage to the SHA context.

10.75.2.372 SHA_MODE_MASK

```
#define SHA_MODE_MASK ((uint8_t)0x07)
```

Mask the bit 0-2.

10.75.2.373 SHA_MODE_READ_CONTEXT

```
#define SHA_MODE_READ_CONTEXT ((uint8_t)0x06)
```

Read current SHA-256 context out of the device.

10.75.2.374 SHA_MODE_SHA256_END

```
#define SHA_MODE_SHA256_END ((uint8_t)0x02)
```

Complete the calculation and return the digest.

10.75.2.375 SHA_MODE_SHA256_PUBLIC

```
#define SHA_MODE_SHA256_PUBLIC ((uint8_t)0x03)
```

Add 64 byte ECC public key in the slot to the SHA context.

10.75.2.376 SHA_MODE_SHA256_START

```
#define SHA_MODE_SHA256_START ((uint8_t)0x00)
```

Initialization, does not accept a message.

10.75.2.377 SHA_MODE_SHA256_UPDATE

```
#define SHA_MODE_SHA256_UPDATE ((uint8_t)0x01)
```

Add 64 bytes in the meesage to the SHA context.

10.75.2.378 SHA_MODE_TARGET_MASK

```
#define SHA_MODE_TARGET_MASK ((uint8_t)0xC0)
```

Resulting digest target location mask.

10.75.2.379 SHA_MODE_WRITE_CONTEXT

```
#define SHA_MODE_WRITE_CONTEXT ((uint8_t)0x07)
```

Restore a SHA-256 context into the device.

10.75.2.380 SHA_RSP_SIZE

```
#define SHA_RSP_SIZE ATCA_RSP_SIZE_32
```

SHA command response packet size.

10.75.2.381 SHA_RSP_SIZE_LONG

```
#define SHA_RSP_SIZE_LONG ATCA_RSP_SIZE_32
```

SHA command response packet size.

10.75.2.382 SHA_RSP_SIZE_SHORT

```
#define SHA_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

SHA command response packet size only status code.

10.75.2.383 SIGN_COUNT

```
#define SIGN_COUNT ATCA_CMD_SIZE_MIN
```

Sign command packet size.

10.75.2.384 SIGN_KEYID_IDX

```
#define SIGN_KEYID_IDX ATCA_PARAM2_IDX
```

Sign command index for key id.

10.75.2.385 SIGN_MODE_EXTERNAL

```
#define SIGN_MODE_EXTERNAL ((uint8_t)0x80)
```

Sign mode bit 7: external.

10.75.2.386 SIGN_MODE_IDX

```
#define SIGN_MODE_IDX ATCA_PARAM1_IDX
```

Sign command index for mode.

10.75.2.387 SIGN_MODE_INCLUDE_SN

```
#define SIGN_MODE_INCLUDE_SN ((uint8_t)0x40)
```

Sign mode bit 6: include serial number.

10.75.2.388 SIGN_MODE_INTERNAL

```
#define SIGN_MODE_INTERNAL ((uint8_t)0x00)
```

Sign mode 0: internal.

10.75.2.389 SIGN_MODE_INVALIDATE

```
#define SIGN_MODE_INVALIDATE ((uint8_t)0x01)
```

Sign mode bit 1: Signature will be used for Verify(Invalidate)

10.75.2.390 SIGN_MODE_MASK

```
#define SIGN_MODE_MASK ((uint8_t)0xE1)
```

Sign mode bits 1 to 4 are 0.

10.75.2.391 SIGN_MODE_SOURCE_MASK

```
#define SIGN_MODE_SOURCE_MASK ((uint8_t)0x20)
```

Sign mode message source mask.

10.75.2.392 SIGN_MODE_SOURCE_MSGDIGBUF

```
#define SIGN_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
```

Sign mode message source is the Message Digest Buffer.

10.75.2.393 SIGN_MODE_SOURCE_TEMPKEY

```
#define SIGN_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

Sign mode message source is TempKey.

10.75.2.394 SIGN_RSP_SIZE

```
#define SIGN_RSP_SIZE ATCA_RSP_SIZE_MAX
```

Sign command response packet size.

10.75.2.395 UPDATE_COUNT

```
#define UPDATE_COUNT ATCA_CMD_SIZE_MIN
```

UpdateExtra command packet size.

10.75.2.396 UPDATE_MODE_DEC_COUNTER

```
#define UPDATE_MODE_DEC_COUNTER ((uint8_t)0x02)
```

UpdateExtra mode: decrement counter.

10.75.2.397 UPDATE_MODE_IDX

```
#define UPDATE_MODE_IDX ATCA_PARAM1_IDX
```

UpdateExtra command index for mode.

10.75.2.398 UPDATE_MODE_SELECTOR

```
#define UPDATE_MODE_SELECTOR ((uint8_t)0x01)
```

UpdateExtra mode update Selector (config byte 85)

10.75.2.399 UPDATE_MODE_USER_EXTRA

```
#define UPDATE_MODE_USER_EXTRA ((uint8_t)0x00)
```

UpdateExtra mode update UserExtra (config byte 84)

10.75.2.400 UPDATE_MODE_USER_EXTRA_ADD

```
#define UPDATE_MODE_USER_EXTRA_ADD UPDATE_MODE_SELECTOR
```

UpdateExtra mode update UserExtraAdd (config byte 85)

10.75.2.401 UPDATE_RSP_SIZE

```
#define UPDATE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

UpdateExtra command response packet size.

10.75.2.402 UPDATE_VALUE_IDX

```
#define UPDATE_VALUE_IDX ATCA_PARAM2_IDX
```

UpdateExtra command index for new value.

10.75.2.403 VERIFY_256_EXTERNAL_COUNT

```
#define VERIFY_256_EXTERNAL_COUNT (135)
```

Verify command packet size for 256-bit key in external mode.

10.75.2.404 VERIFY_256_KEY_SIZE

```
#define VERIFY_256_KEY_SIZE ( 64)
```

Verify key size for 256-bit key.

10.75.2.405 VERIFY_256_SIGNATURE_SIZE

```
#define VERIFY_256_SIGNATURE_SIZE ( 64)
```

Verify signature size for 256-bit key.

10.75.2.406 VERIFY_256_STORED_COUNT

```
#define VERIFY_256_STORED_COUNT ( 71)
```

Verify command packet size for 256-bit key in stored mode.

10.75.2.407 VERIFY_256_VALIDATE_COUNT

```
#define VERIFY_256_VALIDATE_COUNT ( 90)
```

Verify command packet size for 256-bit key in validate mode.

10.75.2.408 VERIFY_283_EXTERNAL_COUNT

```
#define VERIFY_283_EXTERNAL_COUNT (151)
```

Verify command packet size for 283-bit key in external mode.

10.75.2.409 VERIFY_283_KEY_SIZE

```
#define VERIFY_283_KEY_SIZE ( 72)
```

Verify key size for 283-bit key.

10.75.2.410 VERIFY_283_SIGNATURE_SIZE

```
#define VERIFY_283_SIGNATURE_SIZE ( 72)
```

Verify signature size for 283-bit key.

10.75.2.411 VERIFY_283_STORED_COUNT

```
#define VERIFY_283_STORED_COUNT ( 79)
```

Verify command packet size for 283-bit key in stored mode.

10.75.2.412 VERIFY_283_VALIDATE_COUNT

```
#define VERIFY_283_VALIDATE_COUNT ( 98)
```

Verify command packet size for 283-bit key in validate mode.

10.75.2.413 VERIFY_DATA_IDX

```
#define VERIFY_DATA_IDX ( 5)
```

Verify command index for data.

10.75.2.414 VERIFY_KEY_B283

```
#define VERIFY_KEY_B283 ((uint16_t)0x0000)
```

Verify key type: B283.

10.75.2.415 VERIFY_KEY_K283

```
#define VERIFY_KEY_K283 ((uint16_t)0x0001)
```

Verify key type: K283.

10.75.2.416 VERIFY_KEY_P256

```
#define VERIFY_KEY_P256 ((uint16_t)0x0004)
```

Verify key type: P256.

10.75.2.417 VERIFY_KEYID_IDX

```
#define VERIFY_KEYID_IDX ATCA_PARAM2_IDX
```

Verify command index for key id.

10.75.2.418 VERIFY_MODE_EXTERNAL

```
#define VERIFY_MODE_EXTERNAL ((uint8_t)0x02)
```

Verify mode: external.

10.75.2.419 VERIFY_MODE_IDX

```
#define VERIFY_MODE_IDX ATCA_PARAM1_IDX
```

Verify command index for mode.

10.75.2.420 VERIFY_MODE_INVALIDATE

```
#define VERIFY_MODE_INVALIDATE ((uint8_t)0x07)
```

Verify mode: invalidate.

10.75.2.421 VERIFY_MODE_MAC_FLAG

```
#define VERIFY_MODE_MAC_FLAG ((uint8_t)0x80)
```

Verify mode: MAC.

10.75.2.422 VERIFY_MODE_MASK

```
#define VERIFY_MODE_MASK ((uint8_t)0x07)
```

Verify mode bits 3 to 7 are 0.

10.75.2.423 VERIFY_MODE_SOURCE_MASK

```
#define VERIFY_MODE_SOURCE_MASK ((uint8_t)0x20)
```

Verify mode message source mask.

10.75.2.424 VERIFY_MODE_SOURCE_MSGDIGBUF

```
#define VERIFY_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
```

Verify mode message source is the Message Digest Buffer.

10.75.2.425 VERIFY_MODE_SOURCE_TEMPKEY

```
#define VERIFY_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

Verify mode message source is TempKey.

10.75.2.426 VERIFY_MODE_STORED

```
#define VERIFY_MODE_STORED ((uint8_t)0x00)
```

Verify mode: stored.

10.75.2.427 VERIFY_MODE_VALIDATE

```
#define VERIFY_MODE_VALIDATE ((uint8_t)0x03)
```

Verify mode: validate.

10.75.2.428 VERIFY_MODE_VALIDATE_EXTERNAL

```
#define VERIFY_MODE_VALIDATE_EXTERNAL ((uint8_t)0x01)
```

Verify mode: validate external.

10.75.2.429 VERIFY_OTHER_DATA_SIZE

```
#define VERIFY_OTHER_DATA_SIZE ( 19)
```

Verify size of "other data".

10.75.2.430 VERIFY_RSP_SIZE

```
#define VERIFY_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Verify command response packet size.

10.75.2.431 VERIFY_RSP_SIZE_MAC

```
#define VERIFY_RSP_SIZE_MAC ATCA_RSP_SIZE_32
```

Verify command response packet size with validating MAC.

10.75.2.432 WRITE_ADDR_IDX

```
#define WRITE_ADDR_IDX ATCA_PARAM2_IDX
```

Write command index for address.

10.75.2.433 WRITE_MAC_SIZE

```
#define WRITE_MAC_SIZE (32)
```

Write MAC size.

10.75.2.434 WRITE_MAC_VL_IDX

```
#define WRITE_MAC_VL_IDX (37)
```

Write command index for MAC following long data.

10.75.2.435 WRITE_MAC_VS_IDX

```
#define WRITE_MAC_VS_IDX ( 9)
```

Write command index for MAC following short data.

10.75.2.436 WRITE_RSP_SIZE

```
#define WRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Write command response packet size.

10.75.2.437 WRITE_VALUE_IDX

```
#define WRITE_VALUE_IDX ATCA_DATA_IDX
```

Write command index for data.

10.75.2.438 WRITE_ZONE_DATA

```
#define WRITE_ZONE_DATA ((uint8_t)2)
```

Write zone id data.

10.75.2.439 WRITE_ZONE_IDX

```
#define WRITE_ZONE_IDX ATCA_PARAM1_IDX
```

Write command index for zone.

10.75.2.440 WRITE_ZONE_MASK

```
#define WRITE_ZONE_MASK ((uint8_t)0xC3)
```

Write zone bits 2 to 5 are 0.

10.75.2.441 WRITE_ZONE_OTP

```
#define WRITE_ZONE_OTP ((uint8_t)1)
```

Write zone id OTP.

10.75.2.442 WRITE_ZONE_WITH_MAC

```
#define WRITE_ZONE_WITH_MAC ((uint8_t)0x40)
```

Write zone bit 6: write encrypted with MAC.

10.75.3 Function Documentation

10.75.3.1 atAES()

```
ATCA_STATUS atAES (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.2 atCalcCrc()

```
void atCalcCrc (
            ATCAPacket * packet )
```

This function calculates CRC and adds it to the correct offset in the packet data.

Parameters

in	*packet*	Packet to calculate CRC data for

10.75.3.3 atCheckCrc()

```
ATCA_STATUS atCheckCrc (
            const uint8_t * response )
```

This function checks the consistency of a response.

Parameters

in	*response*	pointer to response

Returns

ATCA_SUCCESS on success, otherwise ATCA_RX_CRC_ERROR

10.75.3.4 atCheckMAC()

```
ATCA_STATUS atCheckMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand CheckMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.5 atCounter()

```
ATCA_STATUS atCounter (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Counter method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.75.3.6 atCRC()

```
void atCRC (
            size_t length,
            const uint8_t * data,
            uint8_t * crc_le )
```

Calculates CRC over the given raw data and returns the CRC in little-endian byte order.

Parameters

in	*length*	Size of data not including the CRC byte positions
in	*data*	Pointer to the data over which to compute the CRC
out	*crc_le*	Pointer to the place where the two-bytes of CRC will be returned in little-endian byte order.

10.75.3.7 atDeriveKey()

```
ATCA_STATUS atDeriveKey (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand DeriveKey method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	hasMAC determines if MAC data is present in the packet input

Returns

ATCA_SUCCESS

10.75.3.8 atECDH()

```
ATCA_STATUS atECDH (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand ECDH method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.9 atGenDig()

```
ATCA_STATUS atGenDig (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool is_no_mac_key )
```

ATCACommand Generate Digest method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*is_no_mac_key*	Should be true if GenDig is being run on a slot that has its SlotConfig.NoMac bit set

Returns

ATCA_SUCCESS

10.75.3.10 atGenKey()

```
ATCA_STATUS atGenKey (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Generate Key method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.11 atHMAC()

```
ATCA_STATUS atHMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand HMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.12 atInfo()

```
ATCA_STATUS atInfo (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Info method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.13 atIsECCFamily()

```
bool atIsECCFamily (
            ATCADeviceType device_type )
```

determines if a given device type is an ECC device or a superset of a ECC device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is an ECC family device.

10.75.3.14 atIsSHAFamily()

```
bool atIsSHAFamily (
            ATCADeviceType device_type )
```

determines if a given device type is a SHA device or a superset of a SHA device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is a SHA family device.

10.75.3.15 atKDF()

```
ATCA_STATUS atKDF (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand KDF method.

Parameters

in	*ca_cmd*	Instance
in	*packet*	Pointer to the packet containing the command being built.

Returns

ATCA_SUCCESS

10.75.3.16 atLock()

```
ATCA_STATUS atLock (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Lock method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.17 atMAC()

```
ATCA_STATUS atMAC (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand MAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.18 atNonce()

```
ATCA_STATUS atNonce (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Nonce method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.75.3.19 atPause()

```
ATCA_STATUS atPause (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Pause method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.20 atPrivWrite()

```
ATCA_STATUS atPrivWrite (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand PrivWrite method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.21 atRandom()

```
ATCA_STATUS atRandom (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Random method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.22 atRead()

```
ATCA_STATUS atRead (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Read method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.23 atSecureBoot()

```
ATCA_STATUS atSecureBoot (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand SecureBoot method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.24 atSelfTest()

```
ATCA_STATUS atSelfTest (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.25 atSHA()

```
ATCA_STATUS atSHA (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            uint16_t write_context_size )
```

ATCACommand SHA method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*write_context_size*	the length of the sha write_context data

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.75.3.26 atSign()

```
ATCA_STATUS atSign (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand Sign method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.27 atUpdateExtra()

```
ATCA_STATUS atUpdateExtra (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand UpdateExtra method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

10.75.3.28 atVerify()

```
ATCA_STATUS atVerify (
            ATCADeviceType device_type,
            ATCAPacket * packet )
```

ATCACommand ECDSA Verify method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.75.3.29 atWrite()

```
ATCA_STATUS atWrite (
            ATCADeviceType device_type,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand Write method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	Flag to indicate whether a mac is present or not

Returns

ATCA_SUCCESS

10.75.3.30 isATCAError()

```
ATCA_STATUS isATCAError (
            uint8_t * data )
```

checks for basic error frame in data

Parameters

in	*data*	pointer to received data - expected to be in the form of a CA device response frame

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.76 calib_counter.c File Reference

CryptoAuthLib Basic API methods for Counter command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_counter (ATCADevice device, uint8_t mode, uint16_t counter_id, uint32_t *counter_value)

  Compute the Counter functions.
- ATCA_STATUS calib_counter_increment (ATCADevice device, uint16_t counter_id, uint32_t *counter_value)

  Increments one of the device's monotonic counters.
- ATCA_STATUS calib_counter_read (ATCADevice device, uint16_t counter_id, uint32_t *counter_value)

  Read one of the device's monotonic counters.

10.76.1 Detailed Description

CryptoAuthLib Basic API methods for Counter command.

The Counter command reads or increments the binary count value for one of the two monotonic counters

Note

List of devices that support this command - ATECC508A and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.77 calib_derivekey.c File Reference

CryptoAuthLib Basic API methods for DeriveKey command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_derivekey (ATCADevice device, uint8_t mode, uint16_t target_key, const uint8_t ∗mac)
  Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.

10.77.1 Detailed Description

CryptoAuthLib Basic API methods for DeriveKey command.

The DeriveKey command combines the current value of a key with the nonce stored in TempKey using SHA-256 and derives a new key.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.78 calib_ecdh.c File Reference

CryptoAuthLib Basic API methods for ECDH command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_ecdh_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_↩ t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)
  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS calib_ecdh (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)
  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_enc (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_↩ t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id, const uint8_t num_in[NONCE_NUMIN_SIZE])
  ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS calib_ecdh_ioenc (ATCADevice device, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)
  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS calib_ecdh_tempkey (ATCADevice device, const uint8_t ∗public_key, uint8_t ∗pms)
  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS calib_ecdh_tempkey_ioenc (ATCADevice device, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)
  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.

10.78.1 Detailed Description

CryptoAuthLib Basic API methods for ECDH command.

The ECDH command implements the Elliptic Curve Diffie-Hellman algorithm to combine an internal private key with an external public key to calculate a shared secret.

Note

List of devices that support this command - ATECC508A, ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.78.2 Function Documentation

10.78.2.1 calib_ecdh_enc()

```
ATCA_STATUS calib_ecdh_enc (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * read_key,
            uint16_t read_key_id,
            const uint8_t num_in[NONCE_NUMIN_SIZE] )
```

ECDH command with a private key in a slot and the premaster secret is read from the next slot.

This function only works for even numbered slots with the proper configuration.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*read_key*	Read key for the premaster secret slot (key_id|1).
in	*read_key↔_id*	Read key slot for read_key.
in	*num_in*	20 byte host nonce to inject into Nonce calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.79 calib_execution.c File Reference

Implements an execution handler that executes a given command on a device and returns the results.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_execute_send (ATCADevice device, uint8_t device_address, uint8_t *txdata, uint16↩_t txlength)
- ATCA_STATUS calib_execute_receive (ATCADevice device, uint8_t device_address, uint8_t *rxdata, uint16_t *rxlength)
- ATCA_STATUS calib_execute_command (ATCAPacket *packet, ATCADevice device)

  Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

10.79.1 Detailed Description

Implements an execution handler that executes a given command on a device and returns the results.

This implementation wraps Polling and No polling (simple wait) schemes into a single method and use it across the library. Polling is used by default, however, by defining the ATCA_NO_POLL symbol the code will instead wait an estimated max execution time before requesting the result.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.79.2 Function Documentation

10.79.2.1 calib_execute_command()

```
ATCA_STATUS calib_execute_command (
            ATCAPacket * packet,
            ATCADevice device )
```

Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

Parameters

in,out	*packet*	As input, the packet to be sent. As output, the data buffer in the packet structure will contain the response.
in	*device*	CryptoAuthentication device to send the command to.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.79.2.2 calib_execute_receive()

```
ATCA_STATUS calib_execute_receive (
            ATCADevice device,
            uint8_t device_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

10.79.2.3 calib_execute_send()

```
ATCA_STATUS calib_execute_send (
            ATCADevice device,
            uint8_t device_address,
            uint8_t * txdata,
            uint16_t txlength )
```

10.80 calib_execution.h File Reference

Defines an execution handler that executes a given command on a device and returns the results.

```
#include "atca_status.h"
#include "calib_command.h"
#include "atca_device.h"
#include "atca_config.h"
```

Macros

- #define ATCA_UNSUPPORTED_CMD ((uint16_t)0xFFFF)
- #define CALIB_SWI_FLAG_WAKE 0x00

  flag preceding a command
- #define CALIB_SWI_FLAG_CMD 0x77

  flag preceding a command
- #define CALIB_SWI_FLAG_TX 0x88

  flag requesting a response
- #define CALIB_SWI_FLAG_IDLE 0xBB

  flag requesting to go into Idle mode
- #define CALIB_SWI_FLAG_SLEEP 0xCC

  flag requesting to go into Sleep mode

Functions

- ATCA_STATUS calib_execute_receive (ATCADevice device, uint8_t device_address, uint8_t *rxdata, uint16_t *rxlength)
- ATCA_STATUS calib_execute_command (ATCAPacket *packet, ATCADevice device)

  Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

10.80.1 Detailed Description

Defines an execution handler that executes a given command on a device and returns the results.

The basic flow is to wake the device, send the command, wait/poll for completion, and finally receives the response from the device and does basic checks before returning to caller.

This handler supports the ATSHA and ATECC device family.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.80.2 Macro Definition Documentation

10.80.2.1 ATCA_UNSUPPORTED_CMD

```
#define ATCA_UNSUPPORTED_CMD ((uint16_t)0xFFFF)
```

10.80.2.2 CALIB_SWI_FLAG_CMD

```
#define CALIB_SWI_FLAG_CMD 0x77
```

flag preceding a command

10.80.2.3 CALIB_SWI_FLAG_IDLE

```
#define CALIB_SWI_FLAG_IDLE 0xBB
```

flag requesting to go into Idle mode

10.80.2.4 CALIB_SWI_FLAG_SLEEP

```
#define CALIB_SWI_FLAG_SLEEP 0xCC
```

flag requesting to go into Sleep mode

10.80.2.5 CALIB_SWI_FLAG_TX

```
#define CALIB_SWI_FLAG_TX 0x88
```

flag requesting a response

10.80.2.6 CALIB_SWI_FLAG_WAKE

```
#define CALIB_SWI_FLAG_WAKE 0x00
```

flag preceding a command

10.80.3 Function Documentation

10.80.3.1 calib_execute_command()

```
ATCA_STATUS calib_execute_command (
            ATCAPacket * packet,
            ATCADevice device )
```

Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

Parameters

in,out	*packet*	As input, the packet to be sent. As output, the data buffer in the packet structure will contain the response.
in	*device*	CryptoAuthentication device to send the command to.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.80.3.2 calib_execute_receive()

```
ATCA_STATUS calib_execute_receive (
            ATCADevice device,
            uint8_t device_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

10.81 calib_gendig.c File Reference

CryptoAuthLib Basic API methods for GenDig command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_gendig (ATCADevice device, uint8_t zone, uint16_t key_id, const uint8_t *other_data, uint8_t other_data_size)

    Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.

10.81.1 Detailed Description

CryptoAuthLib Basic API methods for GenDig command.

The GenDig command uses SHA-256 to combine a stored value with the contents of TempKey, which must have been valid prior to the execution of this command.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.82 calib_genkey.c File Reference

CryptoAuthLib Basic API methods for GenKey command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_genkey_base (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_↩ t *other_data, uint8_t *public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS calib_genkey (ATCADevice device, uint16_t key_id, uint8_t *public_key)

  Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS calib_get_pubkey (ATCADevice device, uint16_t key_id, uint8_t *public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS calib_genkey_mac (ATCADevice device, uint8_t *public_key, uint8_t *mac)

  Uses Genkey command to calculate SHA256 digest MAC of combining public key and session key.

10.82.1 Detailed Description

CryptoAuthLib Basic API methods for GenKey command.

The GenKey command is used for creating ECC private keys, generating ECC public keys, and for digest calculations involving public keys.

Note

List of devices that support this command - ATECC108A, ATECC508A, ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.83 calib_helpers.c File Reference

CryptoAuthLib Basic API - Helper Functions to.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_is_slot_locked (ATCADevice device, uint16_t slot, bool *is_locked)

  Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS calib_is_locked (ATCADevice device, uint8_t zone, bool *is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS calib_is_private (ATCADevice device, uint16_t slot, bool *is_private)

  Check if a slot is a private key.

10.83.1 Detailed Description

CryptoAuthLib Basic API - Helper Functions to.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.84 calib_hmac.c File Reference

CryptoAuthLib Basic API methods for HMAC command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_hmac (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t ∗digest)
  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

10.84.1 Detailed Description

CryptoAuthLib Basic API methods for HMAC command.

The HMAC command computes an HMAC/SHA-256 digest using a key stored in the device over a challenge stored in the TempKey register, and/or other information stored within the device.

Note

List of devices that support this command - ATSHA204A, ATECC108A, and ATECC508A . There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.85 calib_info.c File Reference

CryptoAuthLib Basic API methods for Info command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_info_base (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS calib_info (ATCADevice device, uint8_t ∗revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS calib_info_get_latch (ATCADevice device, bool ∗state)

  Use the Info command to get the persistent latch current state for an ATECC608 device.
- ATCA_STATUS calib_info_set_latch (ATCADevice device, bool state)

  Use the Info command to set the persistent latch state for an ATECC608 device.
- ATCA_STATUS calib_info_privkey_valid (ATCADevice device, uint16_t key_id, uint8_t ∗is_valid)

  Use Info command to check ECC Private key stored in key slot is valid or not.

10.85.1 Detailed Description

CryptoAuthLib Basic API methods for Info command.

Info command returns a variety of static and dynamic information about the device and its state. Also is used to control the GPIO pin and the persistent latch.

Note

The ATSHA204A refers to this command as DevRev instead of Info, however, the OpCode and operation is the same.

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A & ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.86 calib_kdf.c File Reference

CryptoAuthLib Basic API methods for KDF command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_kdf (ATCADevice device, uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

  Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.

10.86.1 Detailed Description

CryptoAuthLib Basic API methods for KDF command.

The KDF command implements one of a number of Key Derivation Functions (KDF). Generally this function combines a source key with an input string and creates a result key/digest/array. Three algorithms are currently supported: PRF, HKDF and AES.

Note

List of devices that support this command - ATECC608A/B. Refer to device datasheet for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.87 calib_lock.c File Reference

CryptoAuthLib Basic API methods for Lock command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_lock (ATCADevice device, uint8_t mode, uint16_t summary_crc)

    The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS calib_lock_config_zone (ATCADevice device)

    Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS calib_lock_config_zone_crc (ATCADevice device, uint16_t summary_crc)

    Lock the config zone with summary CRC.
- ATCA_STATUS calib_lock_data_zone (ATCADevice device)

    Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS calib_lock_data_zone_crc (ATCADevice device, uint16_t summary_crc)

    Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS calib_lock_data_slot (ATCADevice device, uint16_t slot)

    Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).
- ATCA_STATUS calib_ecc204_lock_config_slot (ATCADevice device, uint8_t slot, uint16_t summary_crc)

    Use Lock command to lock individual configuration zone slots.
- ATCA_STATUS calib_ecc204_lock_config_zone (ATCADevice device)

    Use lock command to lock complete configuration zone.
- ATCA_STATUS calib_ecc204_lock_data_slot (ATCADevice device, uint8_t slot)

    Use lock command to lock data zone slot.

10.87.1 Detailed Description

CryptoAuthLib Basic API methods for Lock command.

The Lock command prevents future modifications of the Configuration zone, enables configured policies for Data and OTP zones, and can render individual slots read-only regardless of configuration.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.88 calib_mac.c File Reference

CryptoAuthLib Basic API methods for MAC command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_mac (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

10.88.1 Detailed Description

CryptoAuthLib Basic API methods for MAC command.

The MAC command computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device. The output of this command is the digest of this message.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.89 calib_nonce.c File Reference

CryptoAuthLib Basic API methods for Nonce command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_nonce_base (ATCADevice device, uint8_t mode, uint16_t param2, const uint8_t *num_in, uint8_t *rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS calib_nonce (ATCADevice device, const uint8_t *num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_nonce_load (ATCADevice device, uint8_t target, const uint8_t *num_in, uint16_t num_in_size)

  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS calib_nonce_rand (ATCADevice device, const uint8_t *num_in, uint8_t *rand_out)

  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_challenge (ATCADevice device, const uint8_t *num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS calib_challenge_seed_update (ATCADevice device, const uint8_t *num_in, uint8_t *rand_out)

  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS calib_nonce_gen_session_key (ATCADevice device, uint16_t param2, uint8_t *num_in, uint8_t *rand_out)

  Use Nonce command to generate session key for use by a subsequent write command This Mode only supports in ECC204 device.

10.89.1 Detailed Description

CryptoAuthLib Basic API methods for Nonce command.

The Nonce command generates a nonce for use by a subsequent commands of the device by combining an internally generated random number with an input value from the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.90 calib_privwrite.c File Reference

CryptoAuthLib Basic API methods for PrivWrite command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_priv_write (ATCADevice device, uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32], const uint8_t num_in[NONCE_NUMIN_SIZE])
  Executes PrivWrite command, to write externally generated ECC private keys into the device.

10.90.1 Detailed Description

CryptoAuthLib Basic API methods for PrivWrite command.

The PrivWrite command is used to write externally generated ECC private keys into the device.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.90.2 Function Documentation

10.90.2.1 calib_priv_write()

```
ATCA_STATUS calib_priv_write (
            ATCADevice device,
            uint16_t key_id,
            const uint8_t priv_key[36],
            uint16_t write_key_id,
            const uint8_t write_key[32],
            const uint8_t num_in[NONCE_NUMIN_SIZE] )
```

Executes PrivWrite command, to write externally generated ECC private keys into the device.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot to write the external private key into.
in	*priv_key*	External private key (36 bytes) to be written. The first 4 bytes should be zero for P256 curve.
in	*write_key_id*	Write key slot. Ignored if write_key is NULL.
in	*write_key*	Write key (32 bytes). If NULL, perform an unencrypted PrivWrite, which is only available

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.91 calib_random.c File Reference

CryptoAuthLib Basic API methods for Random command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_random (ATCADevice device, uint8_t ∗rand_out)
  Executes Random command, which generates a 32 byte random number from the CryptoAuth device.

10.91.1 Detailed Description

CryptoAuthLib Basic API methods for Random command.

The Random command generates a random number for use by the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.92 calib_read.c File Reference

CryptoAuthLib Basic API methods for Read command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_read_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_↩ t offset, uint8_t *data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS calib_read_serial_number (ATCADevice device, uint8_t *serial_number)

  Executes Read command, which reads the 9 byte serial number of the device from the config zone.
- ATCA_STATUS calib_read_enc (ATCADevice device, uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[NONCE_NUMIN_SIZE])

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS calib_read_config_zone (ATCADevice device, uint8_t *config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS calib_cmp_config_zone (ATCADevice device, uint8_t *config_data, bool *same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS calib_read_sig (ATCADevice device, uint16_t slot, uint8_t *sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS calib_read_pubkey (ATCADevice device, uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS calib_read_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset, uint8_t *data, size_t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.

10.92.1 Detailed Description

CryptoAuthLib Basic API methods for Read command.

The Read command reads words either 4-byte words or 32-byte blocks from one of the memory zones of the device. The data may optionally be encrypted before being returned to the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.92.2 Function Documentation

10.92.2.1 calib_read_enc()

```
ATCA_STATUS calib_read_enc (
            ATCADevice device,
            uint16_t key_id,
            uint8_t block,
            uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[NONCE_NUMIN_SIZE] )
```

Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.

Data zone must be locked for this command to succeed. Can only read 32 byte blocks.

Parameters

in	*device*	Device context pointer
in	*key_id*	The slot ID to read from.
in	*block*	Index of the 32 byte block within the slot to read.
out	*data*	Decrypted (plaintext) data from the read is returned here (32 bytes).
in	*enc_key*	32 byte ReadKey for the slot being read.
in	*enc_key_id*	KeyID of the ReadKey being used.
in	*num_in*	20 byte host nonce to inject into Nonce calculation

returns ATCA_SUCCESS on success, otherwise an error code.

10.93 calib_secureboot.c File Reference

CryptoAuthLib Basic API methods for SecureBoot command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_secureboot (ATCADevice device, uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

    Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS calib_secureboot_mac (ATCADevice device, uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

    Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.

10.93.1 Detailed Description

CryptoAuthLib Basic API methods for SecureBoot command.

The SecureBoot command provides support for secure boot of an external MCU or MPU.

Note

List of devices that support this command - ATECC608A/B. Refer to device datasheet for full details.

Copyright

10.94 calib_selftest.c File Reference

CryptoAuthLib Basic API methods for SelfTest command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_selftest (ATCADevice device, uint8_t mode, uint16_t param2, uint8_t *result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the AT←ECC608 chip.

10.94.1 Detailed Description

CryptoAuthLib Basic API methods for SelfTest command.

The SelfTest command performs a test of one or more of the cryptographic engines within the device.

Note

List of devices that support this command - ATECC608A/B. Refer to device datasheet for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.95 calib_sha.c File Reference

CryptoAuthLib Basic API methods for SHA command.

```
#include "cryptoauthlib.h"
```

Data Structures

- struct hw_sha256_ctx

Functions

- ATCA_STATUS calib_sha_base (ATCADevice device, uint8_t mode, uint16_t length, const uint8_↩
t ∗message, uint8_t ∗data_out, uint16_t ∗data_out_size)

    Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS calib_sha_start (ATCADevice device)

    Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS calib_sha_update (ATCADevice device, const uint8_t ∗message)

    Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS calib_sha_end (ATCADevice device, uint8_t ∗digest, uint16_t length, const uint8_↩
t ∗message)

    Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_read_context (ATCADevice device, uint8_t ∗context, uint16_t ∗context_size)

    Executes SHA command to read the SHA-256 context back. Only for ATECC608 with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS calib_sha_write_context (ATCADevice device, const uint8_t ∗context, uint16_t context_size)

    Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608 with SHA-256 contexts.
- ATCA_STATUS calib_sha (ATCADevice device, uint16_t length, const uint8_t ∗message, uint8_t ∗digest)

    Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_hw_sha2_256_init (ATCADevice device, atca_sha256_ctx_t ∗ctx)

    Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS calib_hw_sha2_256_update (ATCADevice device, atca_sha256_ctx_t ∗ctx, const uint8_↩
t ∗data, size_t data_size)

    Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS calib_hw_sha2_256_finish (ATCADevice device, atca_sha256_ctx_t ∗ctx, uint8_t ∗digest)

    Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS calib_hw_sha2_256 (ATCADevice device, const uint8_t ∗data, size_t data_size, uint8_↩
t ∗digest)

    Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS calib_sha_hmac_init (ATCADevice device, atca_hmac_sha256_ctx_t ∗ctx, uint16_t key_↩
slot)

    Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_update (ATCADevice device, atca_hmac_sha256_ctx_t ∗ctx, const uint8↩
_t ∗data, size_t data_size)

    Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac_finish (ATCADevice device, atca_hmac_sha256_ctx_t ∗ctx, uint8_↩
t ∗digest, uint8_t target)

    Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS calib_sha_hmac (ATCADevice device, const uint8_t ∗data, size_t data_size, uint16_t key↩
_slot, uint8_t ∗digest, uint8_t target)

    Use the SHA command to compute an HMAC/SHA-256 operation.

10.95.1 Detailed Description

CryptoAuthLib Basic API methods for SHA command.

The SHA command Computes a SHA-256 or HMAC/SHA digest for general purpose use by the host system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.96 calib_sign.c File Reference

CryptoAuthLib Basic API methods for Sign command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_sign_base (ATCADevice device, uint8_t mode, uint16_t key_id, uint8_t *signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS calib_sign (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_sign_internal (ATCADevice device, uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t *signature)

  Executes Sign command to sign an internally generated message.
- ATCA_STATUS calib_ecc204_sign (ATCADevice device, uint16_t key_id, const uint8_t *msg, uint8_t *signature)

  Execute sign command to sign the 32 bytes message digest using private key mentioned in slot.

10.96.1 Detailed Description

CryptoAuthLib Basic API methods for Sign command.

The Sign command generates a signature using the private key in slot with ECDSA algorithm.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.97 calib_updateextra.c File Reference

CryptoAuthLib Basic API methods for UpdateExtra command.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS calib_updateextra (ATCADevice device, uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

10.97.1 Detailed Description

CryptoAuthLib Basic API methods for UpdateExtra command.

The UpdateExtra command is used to update the values of the two extra bytes within the Configuration zone after the Configuration zone has been locked.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

10.98 calib_verify.c File Reference

CryptoAuthLib Basic API methods for Verify command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_verify (ATCADevice device, uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS calib_verify_extern (ATCADevice device, const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or TempKey for other devices.
- ATCA_STATUS calib_verify_extern_mac (ATCADevice device, const uint8_t ∗message, const uint8_↩ t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_stored (ATCADevice device, const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608 device or Temp↩ Key for other devices.

- ATCA_STATUS calib_verify_stored_mac (ATCADevice device, const uint8_t *message, const uint8_t *signature, uint16_t key_id, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608.
- ATCA_STATUS calib_verify_validate (ATCADevice device, uint16_t key_id, const uint8_t *signature, const uint8_t *other_data, bool *is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS calib_verify_invalidate (ATCADevice device, uint16_t key_id, const uint8_t *signature, const uint8_t *other_data, bool *is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.

10.98.1 Detailed Description

CryptoAuthLib Basic API methods for Verify command.

The Verify command takes an ECDSA [R,S] signature and verifies that it is correctly generated given an input message digest and public key.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheet for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.99 calib_write.c File Reference

CryptoAuthLib Basic API methods for Write command.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS calib_write (ATCADevice device, uint8_t zone, uint16_t address, const uint8_t *value, const uint8_t *mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS calib_write_zone (ATCADevice device, uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t *data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS calib_write_enc (ATCADevice device, uint16_t key_id, uint8_t block, const uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id, const uint8_t num_in[NONCE_NUMIN_SIZE])

  Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS calib_write_config_zone (ATCADevice device, const uint8_t *config_data)

Executes the Write command, which writes the configuration zone.

- ATCA_STATUS calib_write_pubkey (ATCADevice device, uint16_t slot, const uint8_t *public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS calib_write_bytes_zone (ATCADevice device, uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t *data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS calib_write_config_counter (ATCADevice device, uint16_t counter_id, uint32_t counter_↩ value)

  Initialize one of the monotonic counters in device with a specific value.

10.99.1 Detailed Description

CryptoAuthLib Basic API methods for Write command.

The Write command writes either one 4-byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for a slot, the data may be required to be encrypted by the system prior to being sent to the device

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A/B. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.99.2 Function Documentation

10.99.2.1 calib_write_enc()

```
ATCA_STATUS calib_write_enc (
            ATCADevice device,
            uint16_t key_id,
            uint8_t block,
            const uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id,
            const uint8_t num_in[NONCE_NUMIN_SIZE] )
```

Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.

The function takes clear text bytes and encrypts them for writing over the wire. Data zone must be locked and the slot configuration must be set to encrypted write for the block to be successfully written.

Parameters

in	*device*	Device context pointer
in	*key_id*	Slot ID to write to.
in	*block*	Index of the 32 byte block to write in the slot.
in	*data*	32 bytes of clear text data to be written to the slot
in	*enc_key*	WriteKey to encrypt with for writing
in	*enc_key←_id*	The KeyID of the WriteKey
in	*num_in*	20 byte host nonce to inject into Nonce calculation

returns ATCA_SUCCESS on success, otherwise an error code.

10.100 cryptoauthlib.h File Reference

Single aggregation point for all CryptoAuthLib header files.

```
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "atca_config.h"
#include "atca_compiler.h"
#include "atca_version.h"
#include "atca_status.h"
#include "atca_debug.h"
#include "atca_iface.h"
#include "atca_helpers.h"
#include "hal/atca_hal.h"
#include "atca_cfgs.h"
#include "atca_device.h"
#include "calib/calib_basic.h"
#include "calib/calib_command.h"
#include "calib/calib_aes_gcm.h"
#include "talib/talib_status.h"
#include "talib/talib_basic.h"
#include "atca_basic.h"
```

Macros

- #define ATCA_SHA_SUPPORT 1
- #define ATCA_ATECC608_SUPPORT
- #define ATCA_ECC_SUPPORT 1
- #define ATCA_CA_SUPPORT 1
- #define ATCA_TA_SUPPORT 1
- #define ATCA_SHA256_BLOCK_SIZE (64)
- #define ATCA_SHA256_DIGEST_SIZE (32)
- #define ATCA_AES128_BLOCK_SIZE (16)

- #define ATCA_AES128_KEY_SIZE (16)
- #define ATCA_ECCP256_KEY_SIZE (32)
- #define ATCA_ECCP256_PUBKEY_SIZE (64)
- #define ATCA_ECCP256_SIG_SIZE (64)
- #define ATCA_ZONE_CONFIG ((uint8_t)0x00)
- #define ATCA_ZONE_OTP ((uint8_t)0x01)
- #define ATCA_ZONE_DATA ((uint8_t)0x02)
- #define SHA_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
- #define SHA_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
- #define SHA_MODE_TARGET_OUT_ONLY ((uint8_t)0xC0)
- #define ATCA_STRINGIFY(x) #x
- #define ATCA_TOSTRING(x) ATCA_STRINGIFY(x)
- #define ATCA_TRACE(s, m) atca_trace(s)

10.100.1 Detailed Description

Single aggregation point for all CryptoAuthLib header files.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.100.2 Macro Definition Documentation

10.100.2.1 ATCA_AES128_BLOCK_SIZE

```
#define ATCA_AES128_BLOCK_SIZE (16)
```

10.100.2.2 ATCA_AES128_KEY_SIZE

```
#define ATCA_AES128_KEY_SIZE (16)
```

10.100.2.3 ATCA_ATECC608_SUPPORT

```
#define ATCA_ATECC608_SUPPORT
```

10.100.2.4 ATCA_CA_SUPPORT

```
#define ATCA_CA_SUPPORT 1
```

10.100.2.5 ATCA_ECC_SUPPORT

```
#define ATCA_ECC_SUPPORT 1
```

10.100.2.6 ATCA_ECCP256_KEY_SIZE

```
#define ATCA_ECCP256_KEY_SIZE (32)
```

10.100.2.7 ATCA_ECCP256_PUBKEY_SIZE

```
#define ATCA_ECCP256_PUBKEY_SIZE (64)
```

10.100.2.8 ATCA_ECCP256_SIG_SIZE

```
#define ATCA_ECCP256_SIG_SIZE (64)
```

10.100.2.9 ATCA_SHA256_BLOCK_SIZE

```
#define ATCA_SHA256_BLOCK_SIZE (64)
```

10.100.2.10 ATCA_SHA256_DIGEST_SIZE

```
#define ATCA_SHA256_DIGEST_SIZE (32)
```

10.100.2.11 ATCA_SHA_SUPPORT

```
#define ATCA_SHA_SUPPORT 1
```

Library Configuration File - All build attributes should be included in atca_config.h

10.100.2.12 ATCA_STRINGIFY

```
#define ATCA_STRINGIFY(
              x ) #x
```

10.100.2.13 ATCA_TA_SUPPORT

```
#define ATCA_TA_SUPPORT 1
```

10.100.2.14 ATCA_TOSTRING

```
#define ATCA_TOSTRING(
              x ) ATCA_STRINGIFY(x)
```

10.100.2.15 ATCA_TRACE

```
#define ATCA_TRACE(
              s,
              m ) atca_trace(s)
```

10.100.2.16 ATCA_ZONE_CONFIG

```
#define ATCA_ZONE_CONFIG ((uint8_t)0x00)
```

10.100.2.17 ATCA_ZONE_DATA

```
#define ATCA_ZONE_DATA ((uint8_t)0x02)
```

10.100.2.18 ATCA_ZONE_OTP

```
#define ATCA_ZONE_OTP ((uint8_t)0x01)
```

10.100.2.19 SHA_MODE_TARGET_MSGDIGBUF

```
#define SHA_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
```

Place resulting digest both in Output buffer and Message Digest Buffer

10.100.2.20 SHA_MODE_TARGET_OUT_ONLY

```
#define SHA_MODE_TARGET_OUT_ONLY ((uint8_t)0xC0)
```

Place resulting digest both in Output buffer ONLY

10.100.2.21 SHA_MODE_TARGET_TEMPKEY

```
#define SHA_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

Place resulting digest both in Output buffer and TempKey

10.101 cryptoki.h File Reference

```
#include "pkcs11.h"
```

Macros

- #define PKCS11_HELPER_DLL_IMPORT
- #define PKCS11_HELPER_DLL_EXPORT
- #define PKCS11_HELPER_DLL_LOCAL
- #define PKCS11_API
- #define PKCS11_LOCAL PKCS11_HELPER_DLL_LOCAL
- #define CK_PTR *
- #define CK_DECLARE_FUNCTION(returnType, name) returnType PKCS11_API name
- #define CK_DECLARE_FUNCTION_POINTER(returnType, name) returnType PKCS11_API(*name)
- #define CK_CALLBACK_FUNCTION(returnType, name) returnType(*name)
- #define NULL_PTR 0

10.101.1 Macro Definition Documentation

10.101.1.1 CK_CALLBACK_FUNCTION

```
#define CK_CALLBACK_FUNCTION(
            returnType,
            name ) returnType(*name)
```

10.101.1.2 CK_DECLARE_FUNCTION

```
#define CK_DECLARE_FUNCTION(
              returnType,
              name ) returnType PKCS11_API name
```

10.101.1.3 CK_DECLARE_FUNCTION_POINTER

```
#define CK_DECLARE_FUNCTION_POINTER(
              returnType,
              name ) returnType PKCS11_API(*name)
```

10.101.1.4 CK_PTR

```
#define CK_PTR *
```

10.101.1.5 NULL_PTR

```
#define NULL_PTR 0
```

10.101.1.6 PKCS11_API

```
#define PKCS11_API
```

10.101.1.7 PKCS11_HELPER_DLL_EXPORT

```
#define PKCS11_HELPER_DLL_EXPORT
```

10.101.1.8 PKCS11_HELPER_DLL_IMPORT

```
#define PKCS11_HELPER_DLL_IMPORT
```

10.101.1.9 PKCS11_HELPER_DLL_LOCAL

```
#define PKCS11_HELPER_DLL_LOCAL
```

10.101.1.10 PKCS11_LOCAL

```
#define PKCS11_LOCAL PKCS11_HELPER_DLL_LOCAL
```

10.102 example_cert_chain.c File Reference

```
#include "atcacert/atcacert_def.h"
#include "example_cert_chain.h"
```

Variables

- const atcacert_def_t g_cert_def_0_root
- const atcacert_cert_element_t g_cert_elements_1_signer [ ]
- const uint8_t g_cert_template_1_signer [ ]
- const atcacert_def_t g_cert_def_1_signer
- const uint8_t g_cert_template_2_device [ ]
- const atcacert_def_t g_cert_def_2_device

10.102.1 Variable Documentation

10.102.1.1 g_cert_def_0_root

```
const atcacert_def_t g_cert_def_0_root
```

Initial value:

```
= {
    .type                   = CERTTYPE_X509,
    .template_id            = 0,
    .public_key_dev_loc     = {
        .zone               = DEVZONE_DATA,
        .slot               = 15,
        .is_genkey          = 0,
        .offset             = 0,
        .count              = 72
    }
}
```

10.102.1.2 g_cert_def_1_signer

```
const atcacert_def_t g_cert_def_1_signer
```

10.102.1.3 g_cert_def_2_device

```
const atcacert_def_t g_cert_def_2_device
```

10.102.1.4 g_cert_elements_1_signer

```
const atcacert_cert_element_t g_cert_elements_1_signer[]
```

10.102.1.5 g_cert_template_1_signer

```
const uint8_t g_cert_template_1_signer[]
```

10.102.1.6 g_cert_template_2_device

```
const uint8_t g_cert_template_2_device[]
```

Initial value:

```
= {
    0x30, 0x82, 0x01, 0xa6, 0x30, 0x82, 0x01, 0x4b, 0xa0, 0x03,  0x02, 0x01, 0x02, 0x02, 0x10, 0x41,
    0xa6, 0x8b, 0xe4, 0x36, 0xdd, 0xc3, 0xd8, 0x39, 0xfa, 0xbd,  0xd7, 0x27, 0xd9, 0x74, 0xe7, 0x30,
    0x0a, 0x06, 0x08, 0x2a, 0x86, 0x48, 0xce, 0x3d, 0x04, 0x03,  0x02, 0x30, 0x34, 0x31, 0x14, 0x30,
    0x12, 0x06, 0x03, 0x55, 0x04, 0x0a, 0x0c, 0x0b, 0x45, 0x78,  0x61, 0x6d, 0x70, 0x6c, 0x65, 0x20,
    0x49, 0x6e, 0x63, 0x31, 0x1c, 0x30, 0x1a, 0x06, 0x03, 0x55,  0x04, 0x03, 0x0c, 0x13, 0x45, 0x78,
    0x61, 0x6d, 0x70, 0x6c, 0x65, 0x20, 0x53, 0x69, 0x67, 0x6e,  0x65, 0x72, 0x20, 0x46, 0x46, 0x46,
    0x46, 0x30, 0x20, 0x17, 0x0d, 0x31, 0x37, 0x30, 0x37, 0x31,  0x30, 0x32, 0x30, 0x30, 0x30, 0x30,
    0x30, 0x5a, 0x18, 0x0f, 0x33, 0x30, 0x30, 0x30, 0x31, 0x32,  0x33, 0x31, 0x32, 0x33, 0x35, 0x39,
    0x35, 0x39, 0x5a, 0x30, 0x2f, 0x31, 0x14, 0x30, 0x12, 0x06,  0x03, 0x55, 0x04, 0x0a, 0x0c, 0x0b,
    0x45, 0x78, 0x61, 0x6d, 0x70, 0x6c, 0x65, 0x20, 0x49, 0x6e,  0x63, 0x31, 0x17, 0x30, 0x15, 0x06,
    0x03, 0x55, 0x04, 0x03, 0x0c, 0x0e, 0x45, 0x78, 0x61, 0x6d,  0x70, 0x6c, 0x65, 0x20, 0x44, 0x65,
    0x76, 0x69, 0x63, 0x65, 0x30, 0x59, 0x30, 0x13, 0x06, 0x07,  0x2a, 0x86, 0x48, 0xce, 0x3d, 0x02,
    0x01, 0x06, 0x08, 0x2a, 0x86, 0x48, 0xce, 0x3d, 0x03, 0x01,  0x07, 0x03, 0x42, 0x00, 0x04, 0x96,
    0x27, 0xf1, 0x3e, 0x80, 0xac, 0xf9, 0xd4, 0x12, 0xce, 0x3b,  0x0d, 0x68, 0xf7, 0x4e, 0xb2, 0xc6,
    0x07, 0x35, 0x00, 0xb7, 0x78, 0x5b, 0xac, 0xe6, 0x50, 0x30,  0x54, 0x77, 0x7f, 0xc8, 0x62, 0x21,
    0xce, 0xf2, 0x5a, 0x9a, 0x9e, 0x86, 0x40, 0xc2, 0x29, 0xd6,  0x4a, 0x32, 0x1e, 0xb9, 0x4a, 0x1b,
    0x1c, 0x94, 0xf5, 0x39, 0x88, 0xae, 0xfe, 0x49, 0xcc, 0xfd,  0xbf, 0x8a, 0x0d, 0x34, 0xb8, 0xa3,
    0x42, 0x30, 0x40, 0x30, 0x1d, 0x06, 0x03, 0x55, 0x1d, 0x0e,  0x04, 0x16, 0x04, 0x14, 0x2d, 0xda,
    0x6c, 0x36, 0xd5, 0xa5, 0x5a, 0xce, 0x97, 0x10, 0x3d, 0xbb,  0xaf, 0x9c, 0x66, 0x2a, 0xcd, 0x3e,
    0xe6, 0xcf, 0x30, 0x1f, 0x06, 0x03, 0x55, 0x1d, 0x23, 0x04,  0x18, 0x30, 0x16, 0x80, 0x14, 0xc6,
    0x70, 0xe0, 0x5e, 0x8a, 0x45, 0x0d, 0xb8, 0x2c, 0x00, 0x2a,  0x40, 0x06, 0x39, 0x4c, 0x19, 0x58,
    0x04, 0x35, 0x76, 0x30, 0x0a, 0x06, 0x08, 0x2a, 0x86, 0x48,  0xce, 0x3d, 0x04, 0x03, 0x02, 0x03,
    0x49, 0x00, 0x30, 0x46, 0x02, 0x21, 0x00, 0xe1, 0xfc, 0x00,  0x23, 0xc1, 0x3d, 0x01, 0x3f, 0x22,
    0x31, 0x0b, 0xf0, 0xb8, 0xf4, 0xf4, 0x22, 0xfc, 0x95, 0x96,  0x33, 0x9c, 0xb9, 0x62, 0xb1, 0xfc,
    0x8a, 0x2d, 0xa8, 0x5c, 0xee, 0x67, 0x72, 0x02, 0x21, 0x00,  0xa1, 0x0d, 0x47, 0xe4, 0xfd, 0x0d,
    0x15, 0xd8, 0xde, 0xa1, 0xb5, 0x96, 0x28, 0x4e, 0x7a, 0x0b,  0xbe, 0xcc, 0xec, 0xe8, 0x8e, 0xcc,
    0x7a, 0x31, 0xb3, 0x00, 0x8b, 0xc0, 0x2e, 0x4f, 0x99, 0xc5
}
```

10.103 example_cert_chain.h File Reference

```
#include "atcacert/atcacert_def.h"
```

Variables

- const atcacert_def_t g_cert_def_1_signer
- const atcacert_def_t g_cert_def_2_device

10.103.1 Variable Documentation

10.103.1.1 g_cert_def_1_signer

```
const atcacert_def_t g_cert_def_1_signer [extern]
```

10.103.1.2 g_cert_def_2_device

```
const atcacert_def_t g_cert_def_2_device [extern]
```

10.104 example_pkcs11_config.c File Reference

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11/pkcs11_object.h"
#include "pkcs11/pkcs11_slot.h"
#include "example_cert_chain.h"
```

Macros

- #define pkcs11configLABEL_DEVICE_CERTIFICATE_FOR_TLS "device"
- #define pkcs11configLABEL_JITP_CERTIFICATE "signer"
- #define pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS "device private"
- #define pkcs11configLABEL_DEVICE_PUBLIC_KEY_FOR_TLS "device public"

Functions

- CK_RV pkcs11_config_cert (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr p←Object, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_key (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr p←Object, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_load_objects (pkcs11_slot_ctx_ptr pSlot)

Variables

- const uint8_t atecc608_config [ ]

10.104.1 Macro Definition Documentation

10.104.1.1 pkcs11configLABEL_DEVICE_CERTIFICATE_FOR_TLS

```
#define pkcs11configLABEL_DEVICE_CERTIFICATE_FOR_TLS "device"
```

10.104.1.2 pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS

```
#define pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS "device private"
```

10.104.1.3 pkcs11configLABEL_DEVICE_PUBLIC_KEY_FOR_TLS

```
#define pkcs11configLABEL_DEVICE_PUBLIC_KEY_FOR_TLS "device public"
```

10.104.1.4 pkcs11configLABEL_JITP_CERTIFICATE

```
#define pkcs11configLABEL_JITP_CERTIFICATE "signer"
```

10.104.2 Function Documentation

10.104.2.1 pkcs11_config_cert()

```
CK_RV pkcs11_config_cert (
            pkcs11_lib_ctx_ptr pLibCtx,
            pkcs11_slot_ctx_ptr pSlot,
            pkcs11_object_ptr pObject,
            CK_ATTRIBUTE_PTR pLabel )
```

10.104.2.2 pkcs11_config_key()

```
CK_RV pkcs11_config_key (
            pkcs11_lib_ctx_ptr pLibCtx,
            pkcs11_slot_ctx_ptr pSlot,
            pkcs11_object_ptr pObject,
            CK_ATTRIBUTE_PTR pLabel )
```

10.104.2.3 pkcs11_config_load_objects()

```
CK_RV pkcs11_config_load_objects (
            pkcs11_slot_ctx_ptr pSlot )
```

10.104.3 Variable Documentation

10.104.3.1 atecc608_config

```
const uint8_t atecc608_config[]
```

Initial value:

```
= {
    0x01, 0x23, 0x00, 0x00, 0x00, 0x00, 0x60, 0x01, 0x00, 0x00, 0x00, 0x00, 0xEE, 0x01, 0x01, 0x00,
    0xC0, 0x00, 0x00, 0x01, 0x8F, 0x20, 0xC4, 0x44, 0x87, 0x20, 0x87, 0x20, 0x8F, 0x0F, 0xC4, 0x36,
    0x9F, 0x0F, 0x82, 0x20, 0x0F, 0x0F, 0xC4, 0x44, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
    0x0F, 0x0F, 0x0F, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x33, 0x00, 0x1C, 0x00, 0x13, 0x00, 0x13, 0x00, 0x7C, 0x00, 0x1C, 0x00, 0x3C, 0x00, 0x33, 0x00,
    0x3C, 0x00, 0x3C, 0x00, 0x3C, 0x00, 0x30, 0x00, 0x3C, 0x00, 0x3C, 0x00, 0x3C, 0x00, 0x30, 0x00,
}
```

Standard Configuration Structure for ATECC608 devices

10.105 hal_all_platforms_kit_hidapi.c File Reference

HAL for kit protocol over HID for any platform.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hidapi.h"
#include "atca_hal.h"
#include "hal/kit_protocol.h"
```

Functions

- ATCA_STATUS hal_kit_hid_init (ATCAIface iface, ATCAIfaceCfg *cfg)

  HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

  HAL implementation of Kit HID post init.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_↩ t *rxsize)

  HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_control (ATCAIface iface, uint8_t option, void *param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_kit_hid_release (void *hal_data)

  Close the physical port for HID.

10.105.1 Detailed Description

HAL for kit protocol over HID for any platform.

Copyright

10.106 hal_esp32_i2c.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <driver/i2c.h>
#include "esp_err.h"
#include "esp_log.h"
#include "cryptoauthlib.h"
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL for ASF SERCOM

Macros

- #define I2C0_SDA_PIN 16
- #define I2C0_SCL_PIN 17
- #define I2C1_SDA_PIN 21
- #define I2C1_SCL_PIN 22
- #define ACK_CHECK_EN 0x1
- #define ACK_CHECK_DIS 0x0
- #define ACK_VAL 0x0
- #define NACK_VAL 0x1
- #define LOG_LOCAL_LEVEL ESP_LOG_INFO
- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

Functions

- ATCA_STATUS hal_i2c_change_baud (ATCAIface iface, uint32_t speed)

  method to change the bus speec of I2C
- ATCA_STATUS hal_i2c_init (ATCAIface iface, ATCAIfaceCfg *cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t *txdata, int txlength)

  HAL implementation of I2C send.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t *rxdata, uint16_t *rxlength)

  HAL implementation of I2C receive function.
- ATCA_STATUS hal_i2c_release (void *hal_data)

  manages reference count on given bus and releases resource if no more refences exist
- ATCA_STATUS hal_i2c_control (ATCAIface iface, uint8_t option, void *param, size_t paramlen)

  Perform control operations for the kit protocol.

Variables

- ATCAI2CMaster_t i2c_hal_data [2]
- const char * TAG = "HAL_I2C"
- ATCA_STATUS status

10.106.1 Macro Definition Documentation

10.106.1.1 ACK_CHECK_DIS

```
#define ACK_CHECK_DIS 0x0
```

I2C master will not check ack from slave

10.106.1.2 ACK_CHECK_EN

```
#define ACK_CHECK_EN 0x1
```

I2C master will check ack from slave

10.106.1.3 ACK_VAL

```
#define ACK_VAL 0x0
```

I2C ack value

10.106.1.4 I2C0_SCL_PIN

```
#define I2C0_SCL_PIN 17
```

10.106.1.5 I2C0_SDA_PIN

```
#define I2C0_SDA_PIN 16
```

10.106.1.6 I2C1_SCL_PIN

```
#define I2C1_SCL_PIN 22
```

10.106.1.7 I2C1_SDA_PIN

```
#define I2C1_SDA_PIN 21
```

10.106.1.8 LOG_LOCAL_LEVEL

```
#define LOG_LOCAL_LEVEL ESP_LOG_INFO
```

10.106.1.9 MAX_I2C_BUSES

```
#define MAX_I2C_BUSES 2
```

10.106.1.10 NACK_VAL

```
#define NACK_VAL 0x1
```

I2C nack value

10.106.2 Typedef Documentation

10.106.2.1 ATCAI2CMaster_t

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

10.106.3 Function Documentation

10.106.3.1 hal_i2c_change_baud()

```
ATCA_STATUS hal_i2c_change_baud (
            ATCAIface iface,
            uint32_t speed )
```

method to change the bus speec of I2C

Parameters

in	*iface*	interface on which to change bus speed
in	*speed*	baud rate (typically 100000 or 400000)

10.106.3.2 hal_i2c_control()

```
ATCA_STATUS hal_i2c_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.106.3.3 hal_i2c_init()

```
ATCA_STATUS hal_i2c_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.

HAL implementation of I2C init.

- this HAL implementation assumes you've included the START Twi libraries in your project, otherwise, the HAL layer will not compile because the START TWI drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

this implementation assumes I2C peripheral has been enabled by user. It only initialize an I2C interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.106.3.4 hal_i2c_post_init()

```
ATCA_STATUS hal_i2c_post_init (
            ATCAIface iface )
```

HAL implementation of I2C post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.106.3.5 hal_i2c_receive()

```
ATCA_STATUS hal_i2c_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of I2C receive function.

HAL implementation of I2C receive function for ASF I2C.

HAL implementation of I2C receive function for START I2C.

Parameters

in	*iface*	Device to interact with.
in	*address*	Device address
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
in	*address*	device address
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device word address
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

< I2C master will check ack from slave

< I2C ack value

< I2C nack value

10.106.3.6 hal_i2c_release()

```
ATCA_STATUS hal_i2c_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

manages reference count on given bus and releases resource if no more refernces exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation return ATCA_SUCCESS
in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

10.106.3.7 hal_i2c_send()

```
ATCA_STATUS hal_i2c_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of I2C send.

HAL implementation of I2C send over ASF.

HAL implementation of I2C send over START.

Parameters

in	*iface*	instance
in	*word_address*	device transaction type
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	instance
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	instance
in	*word_address*	device word address
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

< I2C master will check ack from slave

< I2C master will check ack from slave

10.106.4 Variable Documentation

10.106.4.1 i2c_hal_data

```
ATCAI2CMaster_t i2c_hal_data[2]
```

10.106.4.2 status

```
ATCA_STATUS status
```

10.106.4.3 TAG

```
const char* TAG = "HAL_I2C"
```

10.107 hal_esp32_timer.c File Reference

```
#include "atca_hal.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```

Functions

- void ets_delay_us (uint32_t)
- void atca_delay_us (uint32_t delay)
- void atca_delay_ms (uint32_t msec)

10.107.1 Function Documentation

10.107.1.1 atca_delay_ms()

```
void atca_delay_ms (
            uint32_t msec )
```

10.107.1.2 atca_delay_us()

```
void atca_delay_us (
            uint32_t delay )
```

10.107.1.3 ets_delay_us()

```
void ets_delay_us (
            uint32_t  )
```

10.108 hal_freertos.c File Reference

FreeRTOS Hardware/OS Abstration Layer.

```
#include "atca_hal.h"
#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"
```

Macros

- #define ATCA_MUTEX_TIMEOUT portMAX_DELAY

Functions

- void ∗ hal_malloc (size_t size)
- void hal_free (void ∗ptr)
- void hal_rtos_delay_ms (uint32_t ms)

    Timer API implemented at the HAL level.
- ATCA_STATUS hal_create_mutex (void ∗∗ppMutex, char ∗pName)

    Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void ∗pMutex)
- ATCA_STATUS hal_lock_mutex (void ∗pMutex)
- ATCA_STATUS hal_unlock_mutex (void ∗pMutex)

10.108.1 Detailed Description

FreeRTOS Hardware/OS Abstration Layer.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.108.2 Macro Definition Documentation

10.108.2.1 ATCA_MUTEX_TIMEOUT

```
#define ATCA_MUTEX_TIMEOUT portMAX_DELAY
```

10.109 hal_gpio_harmony.c File Reference

ATCA Hardware abstraction layer for 1WIRE or SWI over GPIO.

```
#include "hal_gpio_harmony.h"
```

Functions

- ATCA_STATUS hal_gpio_init (void *hal, ATCAIfaceCfg *cfg)

  initialize an GPIO interface using given config
- ATCA_STATUS hal_gpio_post_init (ATCAIface iface)

  HAL implementation of GPIO post init.
- ATCA_STATUS hal_gpio_device_discovery (ATCAIface iface)

  Discovery Response sequence is used by the master to perform a general bus call to determine if a device is present on the bus.
- ATCA_STATUS hal_gpio_send (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)

  HAL implementation of bit banging send over Harmony.
- ATCA_STATUS hal_gpio_receive (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_t *rxlength)

  HAL implementation of bit banging receive from HARMONY.
- ATCA_STATUS hal_gpio_idle (ATCAIface iface)

  Put the device in idle mode.
- ATCA_STATUS hal_gpio_sleep (ATCAIface iface)

  send sleep command
- ATCA_STATUS hal_gpio_wake (ATCAIface iface)

  send wake token
- ATCA_STATUS hal_gpio_release (void *hal_data)

  releases resource if no more communication

10.109.1 Detailed Description

ATCA Hardware abstraction layer for 1WIRE or SWI over GPIO.

Copyright

10.109.2 Function Documentation

10.109.2.1 hal_gpio_device_discovery()

```
ATCA_STATUS hal_gpio_device_discovery (
            ATCAIface iface )
```

Discovery Response sequence is used by the master to perform a general bus call to determine if a device is present on the bus.

Parameters

in	*iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.2 hal_gpio_idle()

```
ATCA_STATUS hal_gpio_idle (
            ATCAIface iface )
```

Put the device in idle mode.

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.3 hal_gpio_init()

```
ATCA_STATUS hal_gpio_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

initialize an GPIO interface using given config

Parameters

in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.4 hal_gpio_post_init()

```
ATCA_STATUS hal_gpio_post_init (
            ATCAIface iface )
```

HAL implementation of GPIO post init.

Parameters

in	*iface*	ATCAIface instance

Returns

ATCA_SUCCESS

10.109.2.5 hal_gpio_receive()

```
ATCA_STATUS hal_gpio_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of bit banging receive from HARMONY.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.6 hal_gpio_release()

```
ATCA_STATUS hal_gpio_release (
            void * hal_data )
```

releases resource if no more communication

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.7 hal_gpio_send()

```
ATCA_STATUS hal_gpio_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of bit banging send over Harmony.

Parameters

in	*iface*	instance
in	*word_address*	device transaction type
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.8 hal_gpio_sleep()

```
ATCA_STATUS hal_gpio_sleep (
            ATCAIface iface )
```

send sleep command

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.109.2.9 hal_gpio_wake()

```
ATCA_STATUS hal_gpio_wake (
            ATCAIface iface )
```

send wake token

Parameters

in	*iface*	interface to logical device to wakeup

Returns

ATCA_WAKE_SUCCESS on success, otherwise an error code.

10.110 hal_gpio_harmony.h File Reference

ATCA Hardware abstraction layer for SWI over GPIO drivers.

```
#include <stdlib.h>
#include "cryptoauthlib.h"
#include "atca_status.h"
#include "atca_hal.h"
#include "atca_config.h"
```

Macros

Macros for Bit-Banged 1WIRE Timing

Times to drive bits at 230.4 kbps.

- #define tPUP 0
- #define tDSCHG 150
- #define tRESET 96
- #define tRRT 1
- #define tDRR 1
- #define tMSDR 2
- #define tHTSS 150
- #define tDACK 2
- #define tDACK_DLY atca_delay_us(tDACK)
- #define tRRT_DLY atca_delay_ms(tRRT)
- #define tDRR_DLY atca_delay_us(tDRR)
- #define tMSDR_DLY atca_delay_us(tMSDR)
- #define tDSCHG_DLY atca_delay_us(tDSCHG)
- #define tRESET_DLY atca_delay_us(tRESET)
- #define tHTSS_DLY atca_delay_us(tHTSS)
- #define tLOW0_MIN 6
- #define tLOW0_MAX 16
- #define tLOW1_MIN 1
- #define tLOW1_MAX 2
- #define tRCV_MIN 4
- #define tRCV_MAX 6
- #define tBIT_MIN (tLOW0_MIN + tPUP + tRCV_MIN)
- #define tBIT_MAX 75
- #define tWAKEUP 1
- #define tLOW0_TYPICAL (tLOW0_MIN + ((tLOW0_MAX - tLOW0_MIN) / 2))
- #define tLOW1_TYPICAL (tLOW1_MIN + ((tLOW1_MAX - tLOW1_MIN) / 2))
- #define tBIT_TYPICAL (tBIT_MIN + ((tBIT_MAX - tBIT_MIN) / 2))
- #define tLOW0_HDLY atca_delay_us(11)
- #define tRD_HDLY atca_delay_us(1)
- #define tLOW1_HDLY atca_delay_us(1)
- #define tRCV0_HDLY atca_delay_us(11)
- #define tRCV1_HDLY atca_delay_us(14)

- #define tRD_DLY atca_delay_us(1)
- #define tHIGH_SPEED_DLY atca_delay_us(1)
- #define tSWIN_DLY atca_delay_us(1)
- #define tLOW0_DLY atca_delay_us(tLOW0_TYPICAL)
- #define tLOW1_DLY atca_delay_us(tLOW1_TYPICAL)
- #define tBIT_DLY atca_delay_us(tBIT_TYPICAL)
- #define tRCV0_DLY atca_delay_us(tBIT_TYPICAL - tLOW0_TYPICAL)
- #define tRCV1_DLY atca_delay_us(tBIT_TYPICAL - tLOW1_TYPICAL)
- #define send_logic0_1wire(...) send_logic_bit(__VA_ARGS__, ATCA_GPIO_LOGIC_BIT0)
- #define send_logic1_1wire(...) send_logic_bit(__VA_ARGS__, ATCA_GPIO_LOGIC_BIT1)
- #define send_ACK_1wire(...) send_logic0_1wire(__VA_ARGS__)
- #define send_NACK_1wire(...) send_logic1_1wire(__VA_ARGS__)
- #define ATCA_1WIRE_RESET_WORD_ADDR 0x00
- #define ATCA_1WIRE_SLEEP_WORD_ADDR 0x01
- #define ATCA_1WIRE_SLEEP_WORD_ADDR_ALTERNATE 0x02
- #define ATCA_1WIRE_COMMAND_WORD_ADDR 0x03
- #define ATCA_1WIRE_RESPONSE_LENGTH_SIZE 0x01
- #define ATCA_1WIRE_BIT_MASK 0x80
- #define ATCA_GPIO_WRITE 0
- #define ATCA_GPIO_READ 1
- #define ATCA_GPIO_INPUT_DIR 0
- #define ATCA_GPIO_OUTPUT_DIR 1
- #define ATCA_GPIO_LOGIC_BIT0 0
- #define ATCA_GPIO_LOGIC_BIT1 1
- #define ATCA_GPIO_ACK ATCA_GPIO_LOGIC_BIT0
- #define ATCA_GPIO_CLEAR 0
- #define ATCA_GPIO_SET 1
- #define ATCA_MIN_RESPONSE_LENGTH 4

Macros for Bit-Banged SWI Timing

Times to drive bits at 230.4 kbps.

- #define BIT_DELAY_1L atca_delay_us(4)
- #define BIT_DELAY_1H atca_delay_us(4)

    should be 4.34 us, is 4.05us
- #define BIT_DELAY_5 atca_delay_us(26)
- #define BIT_DELAY_7 atca_delay_us(34)
- #define RX_TX_DELAY atca_delay_us(65)
- #define ATCA_SWI_WAKE_WORD_ADDR ((uint8_t)0x00)
- #define ATCA_SWI_CMD_WORD_ADDR ((uint8_t)0x77)
- #define ATCA_SWI_TX_WORD_ADDR ((uint8_t)0x88)
- #define ATCA_SWI_IDLE_WORD_ADDR ((uint8_t)0xBB)
- #define ATCA_SWI_SLEEP_WORD_ADDR ((uint8_t)0xCC)
- #define ATCA_SWI_BIT_MASK 0x01
- enum protocol_type { ATCA_PROTOCOL_1WIRE, ATCA_PROTOCOL_SWI, NO_OF_PROTOCOL }
- enum delay_type {
  LOGIC0_1, LOGIC0_2, LOGIC0_3, LOGIC0_4,
  LOGIC1_1, LOGIC1_2, NO_OF_DELAYS }

10.110.1 Detailed Description

ATCA Hardware abstraction layer for SWI over GPIO drivers.

Copyright

10.110.2 Macro Definition Documentation

10.110.2.1 ATCA_1WIRE_BIT_MASK

```
#define ATCA_1WIRE_BIT_MASK 0x80
```

10.110.2.2 ATCA_1WIRE_COMMAND_WORD_ADDR

```
#define ATCA_1WIRE_COMMAND_WORD_ADDR 0x03
```

10.110.2.3 ATCA_1WIRE_RESET_WORD_ADDR

```
#define ATCA_1WIRE_RESET_WORD_ADDR 0x00
```

10.110.2.4 ATCA_1WIRE_RESPONSE_LENGTH_SIZE

```
#define ATCA_1WIRE_RESPONSE_LENGTH_SIZE 0x01
```

10.110.2.5 ATCA_1WIRE_SLEEP_WORD_ADDR

```
#define ATCA_1WIRE_SLEEP_WORD_ADDR 0x01
```

10.110.2.6 ATCA_1WIRE_SLEEP_WORD_ADDR_ALTERNATE

```
#define ATCA_1WIRE_SLEEP_WORD_ADDR_ALTERNATE 0x02
```

10.110.2.7 ATCA_GPIO_ACK

```
#define ATCA_GPIO_ACK ATCA_GPIO_LOGIC_BIT0
```

10.110.2.8 ATCA_GPIO_CLEAR

```
#define ATCA_GPIO_CLEAR 0
```

10.110.2.9 ATCA_GPIO_INPUT_DIR

```
#define ATCA_GPIO_INPUT_DIR 0
```

10.110.2.10 ATCA_GPIO_LOGIC_BIT0

```
#define ATCA_GPIO_LOGIC_BIT0 0
```

10.110.2.11 ATCA_GPIO_LOGIC_BIT1

```
#define ATCA_GPIO_LOGIC_BIT1 1
```

10.110.2.12 ATCA_GPIO_OUTPUT_DIR

```
#define ATCA_GPIO_OUTPUT_DIR 1
```

10.110.2.13 ATCA_GPIO_READ

```
#define ATCA_GPIO_READ 1
```

10.110.2.14 ATCA_GPIO_SET

```
#define ATCA_GPIO_SET 1
```

10.110.2.15 ATCA_GPIO_WRITE

```
#define ATCA_GPIO_WRITE 0
```

10.110.2.16 ATCA_MIN_RESPONSE_LENGTH

```
#define ATCA_MIN_RESPONSE_LENGTH 4
```

10.110.2.17 ATCA_SWI_BIT_MASK

```
#define ATCA_SWI_BIT_MASK 0x01
```

10.110.2.18 ATCA_SWI_CMD_WORD_ADDR

```
#define ATCA_SWI_CMD_WORD_ADDR ((uint8_t)0x77)
```

10.110.2.19 ATCA_SWI_IDLE_WORD_ADDR

```
#define ATCA_SWI_IDLE_WORD_ADDR ((uint8_t)0xBB)
```

10.110.2.20 ATCA_SWI_SLEEP_WORD_ADDR

```
#define ATCA_SWI_SLEEP_WORD_ADDR ((uint8_t)0xCC)
```

10.110.2.21 ATCA_SWI_TX_WORD_ADDR

```
#define ATCA_SWI_TX_WORD_ADDR ((uint8_t)0x88)
```

10.110.2.22 ATCA_SWI_WAKE_WORD_ADDR

```
#define ATCA_SWI_WAKE_WORD_ADDR ((uint8_t)0x00)
```

SWI WORD Address

10.110.2.23 BIT_DELAY_1H

```
#define BIT_DELAY_1H atca_delay_us(4)
```

should be 4.34 us, is 4.05us

10.110.2.24 BIT_DELAY_1L

```
#define BIT_DELAY_1L atca_delay_us(4)
```

delay macro for width of one pulse (start pulse or zero pulse) should be 4.34 us, is 4.05 us

10.110.2.25 BIT_DELAY_5

```
#define BIT_DELAY_5 atca_delay_us(26)
```

time to keep pin high for five pulses plus stop bit (used to bit-bang CryptoAuth 'zero' bit) should be 26.04 us, is 26.92 us

10.110.2.26 BIT_DELAY_7

```
#define BIT_DELAY_7 atca_delay_us(34)
```

time to keep pin high for seven bits plus stop bit (used to bit-bang CryptoAuth 'one' bit) should be 34.72 us, is 35.13 us

10.110.2.27 RX_TX_DELAY

```
#define RX_TX_DELAY atca_delay_us(65)
```

turn around time when switching from receive to transmit should be 93 us (Setting little less value as there would be other process before these steps)

10.110.2.28 send_ACK_1wire

```
#define send_ACK_1wire(
              ... ) send_logic0_1wire(__VA_ARGS__)
```

10.110.2.29 send_logic0_1wire

```
#define send_logic0_1wire(
              ... ) send_logic_bit(__VA_ARGS__, ATCA_GPIO_LOGIC_BIT0)
```

10.110.2.30 send_logic1_1wire

```
#define send_logic1_1wire(
             ... ) send_logic_bit(__VA_ARGS__, ATCA_GPIO_LOGIC_BIT1)
```

10.110.2.31 send_NACK_1wire

```
#define send_NACK_1wire(
             ... ) send_logic1_1wire(__VA_ARGS__)
```

10.110.2.32 tBIT_DLY

```
#define tBIT_DLY atca_delay_us(tBIT_TYPICAL)
```

10.110.2.33 tBIT_MAX

```
#define tBIT_MAX 75
```

10.110.2.34 tBIT_MIN

```
#define tBIT_MIN (tLOW0_MIN + tPUP + tRCV_MIN)
```

10.110.2.35 tBIT_TYPICAL

```
#define tBIT_TYPICAL (tBIT_MIN + ((tBIT_MAX - tBIT_MIN) / 2 ))
```

10.110.2.36 tDACK

```
#define tDACK 2
```

10.110.2.37 tDACK_DLY

```
#define tDACK_DLY atca_delay_us(tDACK)
```

10.110.2.38 tDRR

```
#define tDRR 1
```

10.110.2.39 tDRR_DLY

```
#define tDRR_DLY atca_delay_us(tDRR)
```

10.110.2.40 tDSCHG

```
#define tDSCHG 150
```

10.110.2.41 tDSCHG_DLY

```
#define tDSCHG_DLY atca_delay_us(tDSCHG)
```

10.110.2.42 tHIGH_SPEED_DLY

```
#define tHIGH_SPEED_DLY atca_delay_us(1)
```

10.110.2.43 tHTSS

```
#define tHTSS 150
```

10.110.2.44 tHTSS_DLY

```
#define tHTSS_DLY atca_delay_us(tHTSS)
```

10.110.2.45 tLOW0_DLY

```
#define tLOW0_DLY atca_delay_us(tLOW0_TYPICAL)
```

10.110.2.46 tLOW0_HDLY

```
#define tLOW0_HDLY atca_delay_us(11)
```

10.110.2.47 tLOW0_MAX

```
#define tLOW0_MAX 16
```

10.110.2.48 tLOW0_MIN

```
#define tLOW0_MIN 6
```

10.110.2.49 tLOW0_TYPICAL

```
#define tLOW0_TYPICAL (tLOW0_MIN + ((tLOW0_MAX - tLOW0_MIN) / 2))
```

10.110.2.50 tLOW1_DLY

```
#define tLOW1_DLY atca_delay_us(tLOW1_TYPICAL)
```

10.110.2.51 tLOW1_HDLY

```
#define tLOW1_HDLY atca_delay_us(1)
```

10.110.2.52 tLOW1_MAX

```
#define tLOW1_MAX 2
```

10.110.2.53 tLOW1_MIN

```
#define tLOW1_MIN 1
```

10.110.2.54 tLOW1_TYPICAL

```
#define tLOW1_TYPICAL (tLOW1_MIN + ((tLOW1_MAX - tLOW1_MIN) / 2))
```

10.110.2.55 tMSDR

```
#define tMSDR 2
```

10.110.2.56 tMSDR_DLY

```
#define tMSDR_DLY atca_delay_us(tMSDR)
```

10.110.2.57 tPUP

```
#define tPUP 0
```

10.110.2.58 tRCV0_DLY

```
#define tRCV0_DLY atca_delay_us(tBIT_TYPICAL - tLOW0_TYPICAL)
```

10.110.2.59 tRCV0_HDLY

```
#define tRCV0_HDLY atca_delay_us(11)
```

10.110.2.60 tRCV1_DLY

```
#define tRCV1_DLY atca_delay_us(tBIT_TYPICAL - tLOW1_TYPICAL)
```

10.110.2.61 tRCV1_HDLY

```
#define tRCV1_HDLY atca_delay_us(14)
```

10.110.2.62 tRCV_MAX

```
#define tRCV_MAX 6
```

10.110.2.63 tRCV_MIN

```
#define tRCV_MIN 4
```

10.110.2.64 tRD_DLY

```
#define tRD_DLY atca_delay_us(1)
```

10.110.2.65 tRD_HDLY

```
#define tRD_HDLY atca_delay_us(1)
```

10.110.2.66 tRESET

```
#define tRESET 96
```

10.110.2.67 tRESET_DLY

```
#define tRESET_DLY atca_delay_us(tRESET)
```

10.110.2.68 tRRT

```
#define tRRT 1
```

10.110.2.69 tRRT_DLY

```
#define tRRT_DLY atca_delay_ms(tRRT)
```

10.110.2.70 tSWIN_DLY

```
#define tSWIN_DLY atca_delay_us(1)
```

10.110.2.71 tWAKEUP

```
#define tWAKEUP 1
```

10.110.3 Enumeration Type Documentation

10.110.3.1 delay_type

```
enum delay_type
```

Enumerator

LOGIC0_1	
LOGIC0_2	
LOGIC0_3	
LOGIC0_4	
LOGIC1_1	
LOGIC1_2	
NO_OF_DELAYS	

10.110.3.2 protocol_type

```
enum protocol_type
```

Enumerator

ATCA_PROTOCOL_1WIRE	
ATCA_PROTOCOL_SWI	
NO_OF_PROTOCOL	

10.111 hal_i2c_harmony.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over Harmony PLIB.

```
#include <string.h>
#include <stdio.h>
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speec of I2C
- ATCA_STATUS hal_i2c_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.111.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over Harmony PLIB.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the Harmony I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

10.112 hal_i2c_start.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include <string.h>
#include <stdio.h>
#include <atmel_start.h>
#include <hal_gpio.h>
#include <hal_delay.h>
#include "hal_i2c_start.h"
#include "atca_start_config.h"
#include "atca_start_iface.h"
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.112.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the START I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.113 hal_i2c_start.h File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include "atmel_start.h"
#include <stdlib.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct i2c_start_instance

Typedefs

- typedef void(∗ start_change_baudrate) (ATCAIface iface, uint32_t speed)
- typedef struct i2c_start_instance i2c_start_instance_t

10.113.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.114 hal_kit_bridge.c File Reference

Kit Bridging HAL for cryptoauthlib. This is not intended to be a zero copy driver. It should work with any interface that confirms to a few basic requirements: a) will accept an arbitrary number of bytes and packetize it if necessary for transmission, b) will block for the duration of the transmit.

```
#include "cryptoauthlib.h"
#include "atca_hal.h"
#include "hal_kit_bridge.h"
```

Functions

- ATCA_STATUS hal_kit_attach_phy (ATCAIfaceCfg *cfg, atca_hal_kit_phy_t *phy)

    Helper function that connects a physical layer context structure that will be used by the kit protocol bridge.
- ATCA_STATUS hal_kit_discover_buses (int busses[ ], int max_buses)

    Request a list of busses from the kit host.
- ATCA_STATUS hal_kit_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int *found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_init (void *hal, ATCAIfaceCfg *cfg)

    HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_post_init (ATCAIface iface)

    HAL implementation of Kit HID post init.
- ATCA_STATUS hal_kit_send (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)

    HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_receive (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_t *rxsize)

    HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_control (ATCAIface iface, uint8_t option)

    Kit Protocol Control.
- ATCA_STATUS hal_kit_release (void *hal_data)

    Close the physical port for HID.

10.114.1 Detailed Description

Kit Bridging HAL for cryptoauthlib. This is not intended to be a zero copy driver. It should work with any interface that confirms to a few basic requirements: a) will accept an arbitrary number of bytes and packetize it if necessary for transmission, b) will block for the duration of the transmit.

Copyright

10.115 hal_kit_bridge.h File Reference

Kit Bridging HAL for cryptoauthlib. This is not intended to be a zero copy driver. It should work with any interface that confirms to a few basic requirements: a) will accept an arbitrary number of bytes and packetize it if necessary for transmission, b) will block for the duration of the transmit.

Macros

- #define HAL_KIT_COMMAND_SEND 0x01
- #define HAL_KIT_COMMAND_RECV 0x02
- #define HAL_KIT_COMMAND_WAKE 0x03
- #define HAL_KIT_COMMAND_IDLE 0x04
- #define HAL_KIT_COMMAND_SLEEP 0x05
- #define HAL_KIT_HEADER_LEN (3)

Functions

- ATCA_STATUS hal_kit_attach_phy (ATCAIfaceCfg *cfg, atca_hal_kit_phy_t *phy)

    Helper function that connects a physical layer context structure that will be used by the kit protocol bridge.

10.115.1 Detailed Description

Kit Bridging HAL for cryptoauthlib. This is not intended to be a zero copy driver. It should work with any interface that confirms to a few basic requirements: a) will accept an arbitrary number of bytes and packetize it if necessary for transmission, b) will block for the duration of the transmit.

Copyright

10.115.2 Macro Definition Documentation

10.115.2.1 HAL_KIT_COMMAND_IDLE

```
#define HAL_KIT_COMMAND_IDLE 0x04
```

10.115.2.2 HAL_KIT_COMMAND_RECV

```
#define HAL_KIT_COMMAND_RECV 0x02
```

10.115.2.3 HAL_KIT_COMMAND_SEND

```
#define HAL_KIT_COMMAND_SEND 0x01
```

10.115.2.4 HAL_KIT_COMMAND_SLEEP

```
#define HAL_KIT_COMMAND_SLEEP 0x05
```

10.115.2.5 HAL_KIT_COMMAND_WAKE

```
#define HAL_KIT_COMMAND_WAKE 0x03
```

10.115.2.6 HAL_KIT_HEADER_LEN

```
#define HAL_KIT_HEADER_LEN (3)
```

10.116 hal_linux.c File Reference

Timer Utility Functions for Linux.

```
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <errno.h>
#include "atca_hal.h"
#include <semaphore.h>
```

Functions

- void hal_delay_us (uint32_t delay)
  This function delays for a number of microseconds.
- void hal_delay_10us (uint32_t delay)
  This function delays for a number of tens of microseconds.
- void hal_delay_ms (uint32_t delay)
  This function delays for a number of milliseconds.
- ATCA_STATUS hal_create_mutex (void **ppMutex, char *pName)
  Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void *pMutex)
- ATCA_STATUS hal_lock_mutex (void *pMutex)
- ATCA_STATUS hal_unlock_mutex (void *pMutex)

10.116.1 Detailed Description

Timer Utility Functions for Linux.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

10.117 hal_linux_i2c_userspace.c File Reference

ATCA Hardware abstraction layer for Linux using I2C.

```
#include <cryptoauthlib.h>
#include <linux/i2c-dev.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "atca_hal.h"
```

Data Structures

- struct atca_i2c_host_s

Typedefs

- typedef struct atca_i2c_host_s atca_i2c_host_t

Functions

- ATCA_STATUS hal_i2c_init (ATCAIface iface, ATCAIfaceCfg *cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t *txdata, int txlength)

  HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t *rxdata, uint16_t *rxlength)

  HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS hal_i2c_control (ATCAIface iface, uint8_t option, void *param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_i2c_release (void *hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.117.1 Detailed Description

ATCA Hardware abstraction layer for Linux using I2C.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.118 hal_linux_spi_userspace.c File Reference

```
#include "cryptoauthlib.h"
#include "atca_hal.h"
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/spi/spidev.h>
```

Data Structures

- struct atca_spi_host_s

Typedefs

- typedef struct atca_spi_host_s atca_spi_host_t

Functions

- ATCA_STATUS hal_spi_open_file (const char ∗dev_name, uint32_t speed, int ∗fd)
  Open and configure the SPI device.
- ATCA_STATUS hal_spi_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)
  HAL implementation of SPI init.
- ATCA_STATUS hal_spi_post_init (ATCAIface iface)
- ATCA_STATUS hal_spi_select (ATCAIface iface)
  HAL implementation to assert the device chip select.
- ATCA_STATUS hal_spi_deselect (ATCAIface iface)
  HAL implementation to deassert the device chip select.
- ATCA_STATUS hal_spi_receive (ATCAIface iface, uint8_t flags, uint8_t ∗rxdata, uint16_t ∗rxlength)
  HAL implementation of SPI receive function.
- ATCA_STATUS hal_spi_send (ATCAIface iface, uint8_t flags, uint8_t ∗txdata, int txlen)
  HAL implementation of SPI send.
- ATCA_STATUS hal_spi_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)
  Perform control operations for the kit protocol.
- ATCA_STATUS hal_spi_release (void ∗hal_data)
  manages reference count on given bus and releases resource if no more refences exist

10.118.1 Typedef Documentation

10.118.1.1 atca_spi_host_t

```
typedef struct atca_spi_host_s atca_spi_host_t
```


10.118.2 Function Documentation

10.118.2.1 hal_spi_control()

```
ATCA_STATUS hal_spi_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the kit protocol.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.2 hal_spi_deselect()

```
ATCA_STATUS hal_spi_deselect (
            ATCAIface iface )
```

HAL implementation to deassert the device chip select.

Parameters

in	*iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.3 hal_spi_init()

```
ATCA_STATUS hal_spi_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

HAL implementation of SPI init.

this implementation assumes SPI peripheral has been enabled by user . It only initialize an SPI interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.4 hal_spi_open_file()

```
ATCA_STATUS hal_spi_open_file (
            const char * dev_name,
            uint32_t speed,
            int * fd )
```

Open and configure the SPI device.

Parameters

in	*dev_name*	File name in the form /dev/spidevX.Y
in	*speed*	Clock speed in Hz
out	*fd*	resulting file descriptor

10.118.2.5 hal_spi_post_init()

```
ATCA_STATUS hal_spi_post_init (
            ATCAIface iface )
```

10.118.2.6 hal_spi_receive()

```
ATCA_STATUS hal_spi_receive (
            ATCAIface iface,
            uint8_t flags,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of SPI receive function.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*len*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.7 hal_spi_release()

```
ATCA_STATUS hal_spi_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.8 hal_spi_select()

```
ATCA_STATUS hal_spi_select (
            ATCAIface iface )
```

HAL implementation to assert the device chip select.

Parameters

in	*iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.118.2.9 hal_spi_send()

```
ATCA_STATUS hal_spi_send (
            ATCAIface iface,
            uint8_t flags,
            uint8_t * txdata,
            int txlen )
```

HAL implementation of SPI send.

Parameters

in	*iface*	instance
in	*word_address*	transaction type
in	*txdata*	data to be send to device
in	*txdata*	pointer to space to bytes to send
in	*len*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119 hal_linux_uart_userspace.c File Reference

ATCA Hardware abstraction layer for Linux using UART.

```
#include "cryptoauthlib.h"
#include "atca_hal.h"
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <termios.h>
```

Data Structures

- struct atca_uart_host_s

Typedefs

- typedef struct atca_uart_host_s atca_uart_host_t

Functions

- ATCA_STATUS hal_uart_open_file (atca_uart_host_t ∗hal_data)

  Open and configure serial COM Uart.
- ATCA_STATUS hal_uart_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  HAL implementation of UART init.
- ATCA_STATUS hal_uart_post_init (ATCAIface iface)

  HAL implementation of UART post init.
- ATCA_STATUS hal_uart_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  HAL implementation of UART send.
- ATCA_STATUS hal_uart_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of UART receive function.
- ATCA_STATUS hal_uart_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations for the UART.
- ATCA_STATUS hal_uart_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.119.1 Detailed Description

ATCA Hardware abstraction layer for Linux using UART.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.119.2 Typedef Documentation

10.119.2.1 atca_uart_host_t

```
typedef struct atca_uart_host_s atca_uart_host_t
```

10.119.3 Function Documentation

10.119.3.1 hal_uart_control()

```
ATCA_STATUS hal_uart_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the UART.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119.3.2 hal_uart_init()

```
ATCA_STATUS hal_uart_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

HAL implementation of UART init.

this implementation assumes UART SERIAL PORT peripheral has been enabled by user . It only initialize an UART interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119.3.3 hal_uart_open_file()

```
ATCA_STATUS hal_uart_open_file (
            atca_uart_host_t * hal_data )
```

Open and configure serial COM Uart.

Parameters

out	*fd*	resulting file descriptor

Returns

ATCA_SUCCESS on success, else an error code

10.119.3.4 hal_uart_post_init()

```
ATCA_STATUS hal_uart_post_init (
            ATCAIface iface )
```

HAL implementation of UART post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119.3.5 hal_uart_receive()

```
ATCA_STATUS hal_uart_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of UART receive function.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119.3.6 hal_uart_release()

```
ATCA_STATUS hal_uart_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.119.3.7 hal_uart_send()

```
ATCA_STATUS hal_uart_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of UART send.

Parameters

in	*iface*	instance
in	*word_address*	transaction type
in	*txdata*	data to be send to device
in	*txdata*	pointer to space to bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.120 hal_sam0_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "hal_sam0_i2c_asf.h"
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

discover any CryptoAuth devices on a given logical bus number

- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

    wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

    idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

    sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

    manages reference count on given bus and releases resource if no more refences exist

10.120.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

10.121 hal_sam0_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct i2c_sam0_instance

Typedefs

- typedef void(∗ sam0_change_baudrate) (ATCAIface iface, uint32_t speed)
- typedef struct i2c_sam0_instance i2c_sam0_instance_t

10.121.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.121.2 Typedef Documentation

10.121.2.1 i2c_sam0_instance_t

```
typedef struct i2c_sam0_instance i2c_sam0_instance_t
```

10.121.2.2 sam0_change_baudrate

```
typedef void(* sam0_change_baudrate) (ATCAIface iface, uint32_t speed)
```

10.122 hal_sam_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAM flexcom & twi I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "cryptoauthlib.h"
#include "hal_sam_i2c_asf.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

    discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

    wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

    idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

    sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

    manages reference count on given bus and releases resource if no more refences exist

10.122.1 Detailed Description

ATCA Hardware abstraction layer for SAM flexcom & twi I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

10.123 hal_sam_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct i2c_sam_instance

Typedefs

- typedef void(∗ sam_change_baudrate) (ATCAIface iface, uint32_t speed)
- typedef struct i2c_sam_instance i2c_sam_instance_t

10.123.1 Detailed Description

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.124 hal_sam_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_us (uint32_t delay)

  This function delays for a number of microseconds.
- void atca_delay_ms (uint32_t ms)

  Timer API for legacy implementations.

10.124.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.125 hal_spi_harmony.c File Reference

ATCA Hardware abstraction layer for SPI over Harmony PLIB.

```
#include <string.h>
#include <stdio.h>
#include "atca_config.h"
#include "cryptoauthlib.h"
#include "atca_hal.h"
#include "atca_device.h"
#include "definitions.h"
#include "talib/talib_defines.h"
#include "talib/talib_fce.h"
```

Functions

- ATCA_STATUS hal_spi_discover_buses (int spi_buses[ ], int max_buses)

  discover spi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_spi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any TA100 devices on a given logical bus number
- ATCA_STATUS hal_spi_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

  initialize an SPI interface using given config
- ATCA_STATUS hal_spi_post_init (ATCAIface iface)

  HAL implementation of SPI post init.
- ATCA_STATUS hal_spi_select (ATCAIface iface)

  HAL implementation to assert the device chip select.
- ATCA_STATUS hal_spi_deselect (ATCAIface iface)

  HAL implementation to deassert the device chip select.
- ATCA_STATUS hal_spi_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  HAL implementation of SPI send over Harmony.
- ATCA_STATUS hal_spi_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of SPI receive function for HARMONY SPI.
- ATCA_STATUS hal_spi_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_spi_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.125.1 Detailed Description

ATCA Hardware abstraction layer for SPI over Harmony PLIB.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical SPI implementation. Part 2 is the Harmony SPI primitives to set up the interface.

Prerequisite: add SERCOM SPI Master Interrupt support to application in Mplab Harmony 3

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.126 hal_swi_bitbang_harmony.c File Reference

ATCA Hardware abstraction layer for SWI bit banging.

```
#include "cryptoauthlib.h"
#include "atca_config.h"
```

Functions

- ATCA_STATUS hal_swi_discover_buses (int swi_buses[ ], int max_buses)

  discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge
- ATCA_STATUS hal_swi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.
- ATCA_STATUS hal_swi_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

  HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

  Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_↩ t ∗rxlength)

  Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

  Send Wake flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

  Send Idle flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

  Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_release (void ∗hal_data)

  Manages reference count on given bus and releases resource if no more reference(s) exist.

10.126.1 Detailed Description

ATCA Hardware abstraction layer for SWI bit banging.

Copyright

10.127 hal_swi_uart.c File Reference

ATCA Hardware abstraction layer for SWI over UART drivers.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS hal_swi_init (ATCAIface iface, ATCAIfaceCfg *cfg)

  initialize an SWI interface using given config
- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

  HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)

  Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_↩ t *rxlength)

  Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

  Send Wake flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

  Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

  Send Idle flag via SWI.
- ATCA_STATUS hal_swi_control (ATCAIface iface, uint8_t option, void *param, size_t paramlen)

  Perform control operations for the kit protocol.
- ATCA_STATUS hal_swi_release (void *hal_data)

  Manages reference count on given bus and releases resource if no more reference(s) exist.

10.127.1 Detailed Description

ATCA Hardware abstraction layer for SWI over UART drivers.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.128 hal_timer_start.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include <hal_delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  This function delays for a number of microseconds.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t ms)

  Timer API for legacy implementations.

10.128.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.129 hal_uart_harmony.c File Reference

ATCA Hardware abstraction layer for SWI uart over Harmony PLIB.

```
#include "atca_config.h"
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS hal_uart_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)

    Initialize an uart interface using given config.
- ATCA_STATUS hal_uart_post_init (ATCAIface iface)

    HAL implementation of SWI post init.
- ATCA_STATUS hal_uart_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)

    Send byte(s) via SWI.
- ATCA_STATUS hal_uart_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_↩ t ∗rxlength)

    Receive byte(s) via SWI.
- ATCA_STATUS hal_uart_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)
- ATCA_STATUS hal_uart_release (void ∗hal_data)

    Manages reference count on given bus and releases resource if no more reference(s) exist.

Variables

- PLIB_SWI_SERIAL_SETUP serial_setup

10.129.1 Detailed Description

ATCA Hardware abstraction layer for SWI uart over Harmony PLIB.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the Harmony UART (ring buffer mode) primitives to set up the interface.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

10.129.2 Function Documentation

10.129.2.1 hal_uart_control()

```
ATCA_STATUS hal_uart_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

10.129.2.2 hal_uart_init()

```
ATCA_STATUS hal_uart_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

Initialize an uart interface using given config.

Parameters

in	*hal*	opaque pointer to HAL data
in	*cfg*	interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.129.2.3 hal_uart_post_init()

```
ATCA_STATUS hal_uart_post_init (
            ATCAIface iface )
```

HAL implementation of SWI post init.

Parameters

in	*iface*	ATCAIface instance

Returns

ATCA_SUCCESS

10.129.2.4 hal_uart_receive()

```
ATCA_STATUS hal_uart_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

Receive byte(s) via SWI.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.129.2.5 hal_uart_release()

```
ATCA_STATUS hal_uart_release (
            void * hal_data )
```

Manages reference count on given bus and releases resource if no more reference(s) exist.

Parameters

in	*hal_data*	opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

10.129.2.6 hal_uart_send()

```
ATCA_STATUS hal_uart_send (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

Send byte(s) via SWI.

Parameters

in	*iface*	interface of the logical device to send data to
in	*word_address*	device transaction type
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS

10.129.3 Variable Documentation

10.129.3.1 serial_setup

```
PLIB_SWI_SERIAL_SETUP serial_setup
```

Initial value:

```
= {
    .parity     = PLIB_SWI_PARITY_NONE,
    .dataWidth  = PLIB_SWI_DATA_WIDTH,
    .stopBits   = PLIB_SWI_STOP_BIT
}
```

10.130 hal_uc3_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "cryptoauthlib.h"
#include "hal_uc3_i2c_asf.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

HAL implementation of I2C post init.

- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t address, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over START.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t address, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for START I2C.
- ATCA_STATUS change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speec of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.130.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.131 hal_uc3_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
#include "twi.h"
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL for ASF SERCOM

Macros

- #define MAX_I2C_BUSES 3

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

    this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- ATCA_STATUS change_i2c_speed (ATCAIface iface, uint32_t speed)

    method to change the bus speec of I2C

10.131.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.132 hal_uc3_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

    This function delays for a number of microseconds.
- void atca_delay_10us (uint32_t delay)

    This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t ms)

    Timer API for legacy implementations.

10.132.1 Detailed Description

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

Prerequisite: add "Delay routines (service)" module to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.133 hal_windows.c File Reference

ATCA Hardware abstraction layer for windows timer functions.

```
#include <windows.h>
#include <math.h>
#include "atca_hal.h"
```

Functions

- void hal_delay_us (uint32_t delay)

  This function delays for a number of microseconds.
- void hal_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void hal_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.
- ATCA_STATUS hal_create_mutex (void ∗∗ppMutex, char ∗pName)

  Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void ∗pMutex)
- ATCA_STATUS hal_lock_mutex (void ∗pMutex)
- ATCA_STATUS hal_unlock_mutex (void ∗pMutex)

10.133.1 Detailed Description

ATCA Hardware abstraction layer for windows timer functions.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.134 hal_windows_kit_uart.c File Reference

ATCA Hardware abstraction layer for Windows using UART.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include "cryptoauthlib.h"
#include "atca_hal.h"
```

Data Structures

- struct atca_uart_host_s

Typedefs

- typedef struct atca_uart_host_s atca_uart_host_t

Functions

- ATCA_STATUS hal_uart_init (ATCAIface iface, ATCAIfaceCfg *cfg)

  HAL implementation of UART init.
- ATCA_STATUS hal_uart_post_init (ATCAIface iface)

  HAL implementation of UART post init.
- ATCA_STATUS hal_uart_send (ATCAIface iface, uint8_t word_address, uint8_t *txdata, int txlength)

  HAL implementation of UART send.
- ATCA_STATUS hal_uart_receive (ATCAIface iface, uint8_t word_address, uint8_t *rxdata, uint16_↩ t *rxlength)

  HAL implementation of UART receive function.
- ATCA_STATUS hal_uart_control (ATCAIface iface, uint8_t option, void *param, size_t paramlen)

  Perform control operations for the UART.
- ATCA_STATUS hal_uart_release (void *hal_data)

  manages reference count on given bus and releases resource if no more refences exist

10.134.1 Detailed Description

ATCA Hardware abstraction layer for Windows using UART.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.134.2 Typedef Documentation

10.134.2.1 atca_uart_host_t

```
typedef struct atca_uart_host_s atca_uart_host_t
```

10.134.3 Function Documentation

10.134.3.1 hal_uart_control()

```
ATCA_STATUS hal_uart_control (
            ATCAIface iface,
            uint8_t option,
            void * param,
            size_t paramlen )
```

Perform control operations for the UART.

Parameters

in	*iface*	Interface to interact with.
in	*option*	Control parameter identifier
in	*param*	Optional pointer to parameter value
in	*paramlen*	Length of the parameter

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.134.3.2 hal_uart_init()

```
ATCA_STATUS hal_uart_init (
            ATCAIface iface,
            ATCAIfaceCfg * cfg )
```

HAL implementation of UART init.

this implementation assumes UART SERIAL PORT peripheral has been enabled by user . It only initialize an UART interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.134.3.3 hal_uart_post_init()

```
ATCA_STATUS hal_uart_post_init (
            ATCAIface iface )
```

HAL implementation of UART post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.134.3.4 hal_uart_receive()

```
ATCA_STATUS hal_uart_receive (
            ATCAIface iface,
            uint8_t word_address,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of UART receive function.

Parameters

in	*iface*	Device to interact with.
in	*word_address*	device transaction type
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.134.3.5 hal_uart_release()

```
ATCA_STATUS hal_uart_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.134.3.6 hal_uart_send()

```
ATCA_STATUS hal_uart_send (
            ATCAIface iface,
```

```
            uint8_t word_address,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of UART send.

Parameters

in	*iface*	instance
in	*word_address*	transaction type
in	*txdata*	data to be send to device
in	*txdata*	pointer to space to bytes to send
in	*len*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.135 io_protection_key.h File Reference

Provides required interface to access IO protection key.

```
#include "atca_status.h"
```

Functions

- ATCA_STATUS io_protection_get_key (uint8_t *io_key)
- ATCA_STATUS io_protection_set_key (uint8_t *io_key)

10.135.1 Detailed Description

Provides required interface to access IO protection key.

Copyright

10.135.2 Function Documentation

10.135.2.1 io_protection_get_key()

```
ATCA_STATUS io_protection_get_key (
            uint8_t * io_key )
```

10.135.2.2 io_protection_set_key()

```
ATCA_STATUS io_protection_set_key (
            uint8_t * io_key )
```

10.136 kit_protocol.c File Reference

Microchip Crypto Auth hardware interface object.

```
#include <stdlib.h>
#include <stdio.h>
#include "atca_compiler.h"
#include "kit_protocol.h"
#include "atca_helpers.h"
```

Macros

- #define KIT_MAX_SCAN_COUNT 4
- #define KIT_MAX_TX_BUF 32

Functions

- char * strnchr (const char *s, size_t count, int c)
- const char * kit_id_from_devtype (ATCADeviceType devtype)
- const char * kit_interface_from_kittype (ATCAKitType kittype)

10.136.1 Detailed Description

Microchip Crypto Auth hardware interface object.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.137 kit_protocol.h File Reference

```
#include "cryptoauthlib.h"
```

Macros

- #define KIT_TX_WRAP_SIZE (10)
- #define KIT_MSG_SIZE (32)
- #define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)

Functions

- ATCA_STATUS kit_init (ATCAIface iface, ATCAIfaceCfg ∗cfg)
- ATCA_STATUS kit_post_init (ATCAIface iface)
- ATCA_STATUS kit_send (ATCAIface iface, uint8_t word_address, uint8_t ∗txdata, int txlength)
- ATCA_STATUS kit_receive (ATCAIface iface, uint8_t word_address, uint8_t ∗rxdata, uint16_t ∗rxsize)
- ATCA_STATUS kit_control (ATCAIface iface, uint8_t option, void ∗param, size_t paramlen)
- ATCA_STATUS kit_release (void ∗hal_data)
- ATCA_STATUS kit_wrap_cmd (const uint8_t ∗txdata, int txlength, char ∗pkitbuf, int ∗nkitbuf, char target)
- ATCA_STATUS kit_parse_rsp (const char ∗pkitbuf, int nkitbuf, uint8_t ∗kitstatus, uint8_t ∗rxdata, int ∗nrxdata)
- ATCA_STATUS kit_wake (ATCAIface iface)
- ATCA_STATUS kit_idle (ATCAIface iface)
- ATCA_STATUS kit_sleep (ATCAIface iface)
- const char ∗ kit_id_from_devtype (ATCADeviceType devtype)
- const char ∗ kit_interface_from_kittype (ATCAKitType kittype)

10.137.1 Detailed Description

Copyright

10.138 license.txt File Reference

Functions

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software. THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUR↩EMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA, OR PROFITS;OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩ NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩ CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩ HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩ URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO T↩ HIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EX↩ TENT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPY↩ RIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSIBILITY OF SUCH this code has the following Version (the "License")
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation files (the "Software")

Variables

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these terms
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER EXPRESS
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR STATUTORY

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS SOFTWARE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON INFRINGEMENT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON MERCHANTABILITY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY SPECIAL
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY PUNITIVE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL LOSS
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO T↩ HIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER CAUSED
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY LAW
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN

ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF ANY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC↩HIP FOR THIS Alan Ott
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC↩HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary forms
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC↩HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without modification
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC↩HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are met
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN

NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright notice

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUD←ING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED INCLUDING
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T←HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED TO
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Mi-

crochip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY INCIDENTAL
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY EXEMPLARY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply

with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN CONTRACT

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT LIABILITY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO T↩HIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EX↩TENT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPY↩RIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSIBILITY OF SUCH this code has the following license
- you may not use this file except in compliance with the License You may obtain a copy of the License at http
- you may not use this file except in compliance with the License You may obtain a copy of the License at

software distributed under the License is distributed on an AS IS BASIS
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY KIND
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the License
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark VanderVoord
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby granted
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of charge
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without restriction
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to use
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to copy
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to modify
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to merge
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↔ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to publish

- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to distribute
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to sublicense
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the Software
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do so
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following conditions
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR IMPLIED
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM
- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI↩ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE

AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR OTHERWISE

- you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITI←ONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR ARISING FROM

10.138.1 Function Documentation

10.138.1.1 DAMAGES()

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES (
            INCLUDING ,
            BUT NOT LIMITED TO,
            PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE,
            DATA ,
            OR PROFITS;OR BUSINESS INTERRUPTION )
```

10.138.1.2 files()

```
you may not use this file except in compliance with the License You may obtain a copy of the
License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES
OR CONDITIONS OF ANY either express or implied See the License for the specific language governing
permissions and limitations under the this code depends on the unity library with the following
Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software
and associated documentation files (
            the "Software"  )
```

10.138.1.3 software()

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
software (
            including open source software )
```

10.138.1.4 TORT()

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (
            INCLUDING NEGLIGENCE OR OTHERWISE )
```

10.138.1.5 Version()

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSI←
BILITY OF SUCH this code has the following Version (
            the "License" )
```

10.138.2 Variable Documentation

10.138.2.1 ANY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF ANY

10.138.2.2 BASIS

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS BASIS

10.138.2.3 CAUSED

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER CAUSED

10.138.2.4 charge

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of charge

10.138.2.5 CLAIM

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM

10.138.2.6 conditions

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following conditions

10.138.2.7 CONTRACT

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF CONT←
RACT

10.138.2.8 copy

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to copy

10.138.2.9 DAMAGE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSI↩
BILITY OF SUCH DAMAGE

Initial value:

```
==============================================================================
If using the Espressif ESP32 I2C driver (lib/hal/hal_esp32_i2c.c)
```

10.138.2.10 DIRECT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY DIRECT

10.138.2.11 distribute

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to distribute

10.138.2.12 EXEMPLARY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←IBUTORS BE LIABLE FOR ANY EXEMPLARY

10.138.2.13 EXPRESS

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER EXPRESS

10.138.2.14 FEES

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES

10.138.2.15 forms

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary forms

10.138.2.16 FROM

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR ARISING FROM

10.138.2.17 granted

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby granted

10.138.2.18 http

you may not use this file except in compliance with the License You may obtain a copy of the License at http

10.138.2.19 IMPLIED

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR IMPLIED

10.138.2.20 INCIDENTAL

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY INCIDENTAL

10.138.2.21 INCLUDING

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED INCLUDING

10.138.2.22 INDIRECT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY INDIRECT

10.138.2.23 INFRINGEMENT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON INFRINGEMENT

10.138.2.24 KIND

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY KIND

10.138.2.25 LAW

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVI↩
SED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY LAW

10.138.2.26 LIABILITY

```
you may not use this file except in compliance with the License You may obtain a copy of the
License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES
OR CONDITIONS OF ANY either express or implied See the License for the specific language governing
permissions and limitations under the this code depends on the unity library with the following
Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software
and associated documentation to deal in the Software without including without limitation the
rights to and or sell copies of the and to permit persons to whom the Software is furnished
to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO
THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY
```

10.138.2.27 license

```
you may not use this file except in compliance with the License You may obtain a copy of the
License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES
OR CONDITIONS OF ANY either express or implied See the License for the specific language governing
permissions and limitations under the this code depends on the unity library with the following
license
```

10.138.2.28 License

```
you may not use this file except in compliance with the License You may obtain a copy of the
License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES
OR CONDITIONS OF ANY either express or implied See the License for the specific language governing
permissions and limitations under the License
```

Initial value:

```
===================================================================================
If using the unit test code in the test folder
```

10.138.2.29 LOSS

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL L↩
OSS
```

10.138.2.30 MERCHANTABILITY

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY

10.138.2.31 merge

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to merge

10.138.2.32 met

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↵ ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↵ OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↵ ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↵ OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are met

10.138.2.33 modification

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↵ ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↵ OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↵ ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↵ OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without modification

10.138.2.34 modify

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to modify

10.138.2.35 notice

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright notice

10.138.2.36 OTHERWISE

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do subject to the following WITHOUT WARRANTY OF ANY EXPRESS OR INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES OR OTHER WHETHER IN AN ACTION OF TORT OR OTHERWISE

10.138.2.37 Ott

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩
ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩
OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Ott

10.138.2.38 publish

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to publish

10.138.2.39 PUNITIVE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY PUNITIVE

10.138.2.40 restriction

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without restriction

10.138.2.41 so

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the and to permit persons to whom the Software is furnished to do so

10.138.2.42 SOFTWARE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩ OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩ ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩ OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE

Initial value:

```
===============================================================================
If using the cross-platform HID driver (lib/hal/hal_all_platforms_kit_hidapi.c)
this code depends on the hidapi library with the following license:
Copyright (c) 2010
```

10.138.2.43 Software

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to and or sell copies of the Software

10.138.2.44 SPECIAL

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY SPECIAL

10.138.2.45 STATUTORY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR STATUTORY

10.138.2.46 sublicense

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to sublicense

10.138.2.47 terms

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these terms

10.138.2.48 TO

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED TO

10.138.2.49 use

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark Greg Williams Permission is hereby free of to any person obtaining a copy of this software and associated documentation to deal in the Software without including without limitation the rights to use

10.138.2.50 VanderVoord

you may not use this file except in compliance with the License You may obtain a copy of the License at software distributed under the License is distributed on an AS IS WITHOUT WARRANTIES OR CONDITIONS OF ANY either express or implied See the License for the specific language governing permissions and limitations under the this code depends on the unity library with the following Mark VanderVoord

10.138.2.51 WARRANTIES

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES

10.139 pkcs11.h File Reference

```
#include "pkcs11t.h"
#include "pkcs11f.h"
```

Data Structures

- struct CK_FUNCTION_LIST

Macros

- #define __PASTE(x, y) x ## y
- #define CK_NEED_ARG_LIST 1
- #define CK_PKCS11_FUNCTION_INFO(name) extern CK_DECLARE_FUNCTION(CK_RV, name)
- #define CK_NEED_ARG_LIST 1
- #define CK_PKCS11_FUNCTION_INFO(name) typedef CK_DECLARE_FUNCTION_POINTER (CK_RV, __PASTE (CK_, name))
- #define CK_PKCS11_FUNCTION_INFO(name) __PASTE(CK_, name) name;

10.139.1 Macro Definition Documentation

10.139.1.1 __PASTE

```
#define __PASTE(
              x,
              y ) x ## y
```

10.139.1.2 CK_NEED_ARG_LIST [1/2]

```
#define CK_NEED_ARG_LIST 1
```

10.139.1.3 CK_NEED_ARG_LIST [2/2]

```
#define CK_NEED_ARG_LIST 1
```

10.139.1.4 CK_PKCS11_FUNCTION_INFO [1/3]

```
#define CK_PKCS11_FUNCTION_INFO(
              name )   extern CK_DECLARE_FUNCTION(CK_RV, name)
```

10.139.1.5 CK_PKCS11_FUNCTION_INFO [2/3]

```
#define CK_PKCS11_FUNCTION_INFO(
              name )   typedef CK_DECLARE_FUNCTION_POINTER (CK_RV, __PASTE (CK_, name))
```

10.139.1.6 CK_PKCS11_FUNCTION_INFO [3/3]

```
#define CK_PKCS11_FUNCTION_INFO(
              name ) __PASTE(CK_, name) name;
```

10.140 pkcs11_attrib.c File Reference

PKCS11 Library Object Attributes Handling.

```
#include "pkcs11_config.h"
#include "pkcs11_attrib.h"
#include "cryptoauthlib.h"
```

Functions

- CK_RV pkcs11_attrib_fill (CK_ATTRIBUTE_PTR pAttribute, const CK_VOID_PTR pData, const CK_ULONG ulSize)
  Perform the nessasary checks and copy data into an attribute structure.
- CK_RV pkcs11_attrib_value (CK_ATTRIBUTE_PTR pAttribute, const CK_ULONG ulValue, const CK_ULONG ulSize)
  Helper function to write a numerical value to an attribute buffer.
- CK_RV pkcs11_attrib_false (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_true (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_empty (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)

10.140.1 Detailed Description

PKCS11 Library Object Attributes Handling.

Copyright

10.141 pkcs11_attrib.h File Reference

PKCS11 Library Object Attribute Handling.

```
#include "cryptoki.h"
```

Data Structures

- struct _pkcs11_attrib_model

Typedefs

- typedef CK_RV(* attrib_f) (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- typedef struct _pkcs11_attrib_model pkcs11_attrib_model
- typedef struct _pkcs11_attrib_model * pkcs11_attrib_model_ptr

Functions

- CK_RV pkcs11_attrib_fill (CK_ATTRIBUTE_PTR pAttribute, const CK_VOID_PTR pData, const CK_ULONG ulSize)

  Perform the nessasary checks and copy data into an attribute structure.
- CK_RV pkcs11_attrib_value (CK_ATTRIBUTE_PTR pAttribute, const CK_ULONG ulValue, const CK_ULONG ulSize)

  Helper function to write a numerical value to an attribute buffer.
- CK_RV pkcs11_attrib_false (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_true (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_attrib_empty (const CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)

10.141.1 Detailed Description

PKCS11 Library Object Attribute Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.141.2 Typedef Documentation

10.141.2.1 attrib_f

```
typedef CK_RV(* attrib_f) (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
```

Populate an attribute based on the "object"

10.141.2.2 pkcs11_attrib_model

```
typedef struct _pkcs11_attrib_model pkcs11_attrib_model
```

10.141.2.3 pkcs11_attrib_model_ptr

```
typedef struct _pkcs11_attrib_model * pkcs11_attrib_model_ptr
```

10.142 pkcs11_cert.c File Reference

PKCS11 Library Certificate Handling.

```
#include "cryptoauthlib.h"
#include "atcacert/atcacert_def.h"
#include "atcacert/atcacert_client.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_token.h"
#include "pkcs11_cert.h"
#include "pkcs11_os.h"
#include "pkcs11_util.h"
```

Functions

- CK_RV pkcs11_cert_get_encoded (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_subject (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_subject_key_id (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_authority_key_id (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_get_trusted_flag (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_cert_x509_write (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)

Variables

- const pkcs11_attrib_model pkcs11_cert_x509public_attributes [ ]
- const CK_ULONG pkcs11_cert_x509public_attributes_count = sizeof(pkcs11_cert_x509public_attributes) / sizeof(pkcs11_cert_x509public_attributes [0])
- const pkcs11_attrib_model pkcs11_cert_wtlspublic_attributes [ ]
- const CK_ULONG pkcs11_cert_wtlspublic_attributes_count = sizeof(pkcs11_cert_wtlspublic_attributes) / sizeof(pkcs11_cert_wtlspublic_attributes [0])
- const pkcs11_attrib_model pkcs11_cert_x509_attributes [ ]
- const CK_ULONG pkcs11_cert_x509_attributes_count = sizeof(pkcs11_cert_x509_attributes) / sizeof(pkcs11_cert_x509_attributes [0])

10.142.1 Detailed Description

PKCS11 Library Certificate Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.143 pkcs11_cert.h File Reference

PKCS11 Library Certificate Handling.

```
#include "pkcs11_object.h"
```

Functions

- CK_RV pkcs11_cert_x509_write (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)

Variables

- const pkcs11_attrib_model pkcs11_cert_x509public_attributes [ ]
- const CK_ULONG pkcs11_cert_x509public_attributes_count
- const pkcs11_attrib_model pkcs11_cert_wtlspublic_attributes [ ]
- const CK_ULONG pkcs11_cert_wtlspublic_attributes_count
- const pkcs11_attrib_model pkcs11_cert_x509_attributes [ ]
- const CK_ULONG pkcs11_cert_x509_attributes_count

10.143.1 Detailed Description

PKCS11 Library Certificate Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.144 pkcs11_config.c File Reference

PKCS11 Library Configuration.

```
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "cryptoauthlib.h"
#include "pkcs11_slot.h"
#include "pkcs11_object.h"
#include "pkcs11_key.h"
#include "pkcs11_cert.h"
#include "pkcs11_os.h"
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
```

Functions

- void pkcs11_config_init_private (pkcs11_object_ptr pObject, char ∗label, size_t len)
- void pkcs11_config_init_public (pkcs11_object_ptr pObject, char ∗label, size_t len)
- void pkcs11_config_init_secret (pkcs11_object_ptr pObject, char ∗label, size_t len, uint8_t keylen)
- void pkcs11_config_init_cert (pkcs11_object_ptr pObject, char ∗label, size_t len)
- CK_RV pkcs11_config_cert (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr pObject, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_key (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr pObject, CK_ATTRIBUTE_PTR pLabel)
- CK_RV pkcs11_config_remove_object (pkcs11_lib_ctx_ptr pLibCtx, pkcs11_slot_ctx_ptr pSlot, pkcs11_object_ptr pObject)
- CK_RV pkcs11_config_load_objects (pkcs11_slot_ctx_ptr slot_ctx)
- CK_RV pkcs11_config_load (pkcs11_slot_ctx_ptr slot_ctx)

10.144.1 Detailed Description

PKCS11 Library Configuration.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.145 pkcs11_debug.c File Reference

PKCS11 Library Debugging.

```
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_os.h"
#include "atca_helpers.h"
```

10.145.1 Detailed Description

PKCS11 Library Debugging.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.146 pkcs11_debug.h File Reference

PKCS11 Library Debugging.

```
#include "pkcs11_config.h"
```

Macros

- #define PKCS11_DEBUG_NOFILE(...)
- #define PKCS11_DEBUG(...)
- #define PKCS11_DEBUG_RETURN(x) { return x; }
- #define pkcs11_debug_attributes(x, y)

10.146.1 Detailed Description

PKCS11 Library Debugging.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.146.2 Macro Definition Documentation

10.146.2.1 PKCS11_DEBUG

```
#define PKCS11_DEBUG(
              ... )
```

10.146.2.2 pkcs11_debug_attributes

```
#define pkcs11_debug_attributes(
              x,
              y )
```

10.146.2.3 PKCS11_DEBUG_NOFILE

```
#define PKCS11_DEBUG_NOFILE(
              ... )
```

10.146.2.4 PKCS11_DEBUG_RETURN

```
#define PKCS11_DEBUG_RETURN(
              x ) { return x; }
```

10.147 pkcs11_digest.c File Reference

```
#include "pkcs11_init.h"
#include "pkcs11_digest.h"
#include "pkcs11_object.h"
#include "pkcs11_session.h"
#include "pkcs11_util.h"
#include "cryptoauthlib.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Functions

- CK_RV pkcs11_digest_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism)

  Initializes a message-digesting operation using the specified mechanism in the specified session.
- CK_RV pkcs11_digest (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Digest the specified data in a one-pass operation and return the resulting digest.
- CK_RV pkcs11_digest_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ul←PartLen)

  Continues a multiple-part digesting operation.
- CK_RV pkcs11_digest_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Finishes a multiple-part digesting operation.

10.147.1 Function Documentation

10.147.1.1 pkcs11_digest()

```
CK_RV pkcs11_digest (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Digest the specified data in a one-pass operation and return the resulting digest.

10.147.1.2 pkcs11_digest_final()

```
CK_RV pkcs11_digest_final (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Finishes a multiple-part digesting operation.

10.147.1.3 pkcs11_digest_init()

```
CK_RV pkcs11_digest_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism )
```

Initializes a message-digesting operation using the specified mechanism in the specified session.

10.147.1.4 pkcs11_digest_update()

```
CK_RV pkcs11_digest_update (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part digesting operation.

10.148 pkcs11_digest.h File Reference

PKCS11 Library Digest (SHA256) Handling.

```
#include "cryptoki.h"
```

Functions

- CK_RV pkcs11_digest_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism)

  Initializes a message-digesting operation using the specified mechanism in the specified session.
- CK_RV pkcs11_digest (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Digest the specified data in a one-pass operation and return the resulting digest.
- CK_RV pkcs11_digest_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ul←PartLen)

  Continues a multiple-part digesting operation.
- CK_RV pkcs11_digest_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

  Finishes a multiple-part digesting operation.

10.148.1 Detailed Description

PKCS11 Library Digest (SHA256) Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.148.2 Function Documentation

10.148.2.1 pkcs11_digest()

```
CK_RV pkcs11_digest (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pData,
            CK_ULONG ulDataLen,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Digest the specified data in a one-pass operation and return the resulting digest.

10.148.2.2 pkcs11_digest_final()

```
CK_RV pkcs11_digest_final (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pDigest,
            CK_ULONG_PTR pulDigestLen )
```

Finishes a multiple-part digesting operation.

10.148.2.3 pkcs11_digest_init()

```
CK_RV pkcs11_digest_init (
            CK_SESSION_HANDLE hSession,
            CK_MECHANISM_PTR pMechanism )
```

Initializes a message-digesting operation using the specified mechanism in the specified session.

10.148.2.4 pkcs11_digest_update()

```
CK_RV pkcs11_digest_update (
            CK_SESSION_HANDLE hSession,
            CK_BYTE_PTR pPart,
            CK_ULONG ulPartLen )
```

Continues a multiple-part digesting operation.

10.149 pkcs11_encrypt.c File Reference

PKCS11 Library Encrypt Support.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11_encrypt.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_object.h"
#include "pkcs11_session.h"
#include "pkcs11_util.h"
```

Functions

- CK_RV pkcs11_encrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_encrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
- CK_RV pkcs11_encrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
- CK_RV pkcs11_encrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

    Finishes a multiple-part encryption operation.
- CK_RV pkcs11_decrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_decrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)
- CK_RV pkcs11_decrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)
- CK_RV pkcs11_decrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

    Finishes a multiple-part decryption operation.

10.149.1 Detailed Description

PKCS11 Library Encrypt Support.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.150 pkcs11_encrypt.h File Reference

PKCS11 Library AES Support.

```
#include "pkcs11.h"
```

Functions

- CK_RV pkcs11_encrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_encrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
- CK_RV pkcs11_encrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)
- CK_RV pkcs11_encrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

    Finishes a multiple-part encryption operation.
- CK_RV pkcs11_decrypt_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_decrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)
- CK_RV pkcs11_decrypt_update (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pDataLen)
- CK_RV pkcs11_decrypt_final (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG_PTR pDataLen)

    Finishes a multiple-part decryption operation.

10.150.1 Detailed Description

PKCS11 Library AES Support.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.151 pkcs11_find.c File Reference

PKCS11 Library Object Find/Searching.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_os.h"
#include "pkcs11_slot.h"
#include "pkcs11_session.h"
#include "pkcs11_find.h"
#include "pkcs11_util.h"
```

Functions

- CK_RV pkcs11_find_init (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_find_continue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount)
- CK_RV pkcs11_find_finish (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_find_get_attribute (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

10.151.1 Detailed Description

PKCS11 Library Object Find/Searching.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.152 pkcs11_find.h File Reference

PKCS11 Library Object Find/Searching.

```
#include "cryptoki.h"
#include "pkcs11_object.h"
```

Functions

- CK_RV pkcs11_find_init (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_find_continue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount)
- CK_RV pkcs11_find_finish (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_find_get_attribute (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

10.152.1 Detailed Description

PKCS11 Library Object Find/Searching.

Copyright

10.153 pkcs11_info.c File Reference

PKCS11 Library Information Functions.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11_init.h"
#include "pkcs11_slot.h"
#include "pkcs11_session.h"
#include "pkcs11_util.h"
#include <stdio.h>
```

Functions

- CK_RV pkcs11_get_lib_info (CK_INFO_PTR pInfo)

  Obtains general information about Cryptoki.

Variables

- const char pkcs11_lib_manufacturer_id [ ] = "Microchip Technology Inc"
- const char pkcs11_lib_description [ ] = "Cryptoauthlib PKCS11 Interface"

10.153.1 Detailed Description

PKCS11 Library Information Functions.

Copyright

10.154 pkcs11_info.h File Reference

PKCS11 Library Information Functions.

```
#include "cryptoki.h"
```

Functions

- CK_RV pkcs11_get_lib_info (CK_INFO_PTR pInfo)
  Obtains general information about Cryptoki.

Variables

- const char pkcs11_lib_manufacturer_id [ ]
- const char pkcs11_lib_description [ ]

10.154.1 Detailed Description

PKCS11 Library Information Functions.

Copyright

10.155 pkcs11_init.c File Reference

PKCS11 Library Init/Deinit.

```
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_os.h"
#include "pkcs11_slot.h"
#include "pkcs11_object.h"
#include "pkcs11_session.h"
#include "cryptoauthlib.h"
```

Functions

- pkcs11_lib_ctx_ptr pkcs11_get_context (void)
  Retrieve the current library context.
- CK_RV pkcs11_lock_context (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_unlock_context (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_init_check (pkcs11_lib_ctx_ptr ∗ppContext, CK_BBOOL lock)
  Check if the library is initialized properly.
- CK_RV pkcs11_init (CK_C_INITIALIZE_ARGS_PTR pInitArgs)
  Initializes the PKCS11 API Library for Cryptoauthlib.
- CK_RV pkcs11_deinit (CK_VOID_PTR pReserved)

10.155.1 Detailed Description

PKCS11 Library Init/Deinit.

Copyright

10.156 pkcs11_init.h File Reference

PKCS11 Library Initialization & Context.

```
#include "cryptoki.h"
#include "pkcs11_config.h"
```

Data Structures

- struct _pkcs11_lib_ctx

Typedefs

- typedef struct _pkcs11_lib_ctx pkcs11_lib_ctx
- typedef struct _pkcs11_lib_ctx * pkcs11_lib_ctx_ptr

Functions

- CK_RV pkcs11_init (CK_C_INITIALIZE_ARGS_PTR pInitArgs)

  Initializes the PKCS11 API Library for Cryptoauthlib.
- CK_RV pkcs11_deinit (CK_VOID_PTR pReserved)
- CK_RV pkcs11_init_check (pkcs11_lib_ctx_ptr *ppContext, CK_BBOOL lock)

  Check if the library is initialized properly.
- pkcs11_lib_ctx_ptr pkcs11_get_context (void)

  Retrieve the current library context.
- CK_RV pkcs11_lock_context (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_unlock_context (pkcs11_lib_ctx_ptr pContext)

10.156.1 Detailed Description

PKCS11 Library Initialization & Context.

Copyright

10.156.2 Typedef Documentation

10.156.2.1 pkcs11_lib_ctx

```
typedef struct _pkcs11_lib_ctx pkcs11_lib_ctx
```

Library Context

10.156.2.2 pkcs11_lib_ctx_ptr

```
typedef struct _pkcs11_lib_ctx * pkcs11_lib_ctx_ptr
```

10.157 pkcs11_key.c File Reference

PKCS11 Library Key Object Handling.

```
#include "cryptoauthlib.h"
#include "crypto/atca_crypto_sw_sha1.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_token.h"
#include "pkcs11_attrib.h"
#include "pkcs11_key.h"
#include "pkcs11_session.h"
#include "pkcs11_slot.h"
#include "pkcs11_util.h"
#include "pkcs11_os.h"
```

Functions

- CK_RV pkcs11_key_write (CK_VOID_PTR pSession, CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_key_generate (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)
- CK_RV pkcs11_key_generate_pair (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR pPrivateKeyTemplate, CK_ULONG ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)
- CK_RV pkcs11_key_derive (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hBaseKey, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

Variables

- const pkcs11_attrib_model pkcs11_key_public_attributes [ ]
- const CK_ULONG pkcs11_key_public_attributes_count = sizeof(pkcs11_key_public_attributes) / sizeof(pkcs11_key_public_attributes [0])
- const pkcs11_attrib_model pkcs11_key_ec_public_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_private_attributes [ ]
- const CK_ULONG pkcs11_key_private_attributes_count = sizeof(pkcs11_key_private_attributes) / sizeof(pkcs11_key_private_attributes [0])
- const pkcs11_attrib_model pkcs11_key_rsa_private_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_ec_private_attributes [ ]
- const pkcs11_attrib_model pkcs11_key_secret_attributes [ ]
- const CK_ULONG pkcs11_key_secret_attributes_count = sizeof(pkcs11_key_secret_attributes) / sizeof(pkcs11_key_secret_attributes [0])

10.157.1 Detailed Description

PKCS11 Library Key Object Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.158 pkcs11_key.h File Reference

PKCS11 Library Object Handling.

```
#include "pkcs11_object.h"
```

Functions

- CK_RV pkcs11_key_write (CK_VOID_PTR pSession, CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR p↵ Attribute)
- CK_RV pkcs11_key_generate (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)
- CK_RV pkcs11_key_generate_pair (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p↵ Mechanism, CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR pPrivateKeyTemplate, CK_ULONG ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)
- CK_RV pkcs11_key_derive (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hBaseKey, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

Variables

- const pkcs11_attrib_model pkcs11_key_public_attributes [ ]
- const CK_ULONG pkcs11_key_public_attributes_count
- const pkcs11_attrib_model pkcs11_key_private_attributes [ ]
- const CK_ULONG pkcs11_key_private_attributes_count
- const pkcs11_attrib_model pkcs11_key_secret_attributes [ ]
- const CK_ULONG pkcs11_key_secret_attributes_count

10.158.1 Detailed Description

PKCS11 Library Object Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.159 pkcs11_main.c File Reference

PKCS11 Basic library redirects based on the 2.40 specification http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html.

```
#include "cryptoki.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_encrypt.h"
#include "pkcs11_init.h"
#include "pkcs11_info.h"
#include "pkcs11_slot.h"
#include "pkcs11_mech.h"
#include "pkcs11_session.h"
#include "pkcs11_token.h"
#include "pkcs11_find.h"
#include "pkcs11_object.h"
#include "pkcs11_signature.h"
#include "pkcs11_digest.h"
#include "pkcs11_key.h"
```

Functions

- CK_RV C_Initialize (CK_VOID_PTR pInitArgs)

  Initializes Cryptoki library NOTES: If pInitArgs is a non-NULL_PTR is must dereference to a CK_C_INITIALIZE_ARGS structure.
- CK_RV C_Finalize (CK_VOID_PTR pReserved)

  Clean up miscellaneous Cryptoki-associated resources.
- CK_RV C_GetInfo (CK_INFO_PTR pInfo)

  Obtains general information about Cryptoki.
- CK_RV C_GetFunctionList (CK_FUNCTION_LIST_PTR_PTR ppFunctionList)

  Obtains entry points of Cryptoki library functions.
- CK_RV C_GetSlotList (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pulCount)

  Obtains a list of slots in the system.
- CK_RV C_GetSlotInfo (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo)

  Obtains information about a particular slot.
- CK_RV C_GetTokenInfo (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo)

  Obtains information about a particular token.
- CK_RV C_GetMechanismList (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount)

  Obtains a list of mechanisms supported by a token (in a slot)

- CK_RV C_GetMechanismInfo (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo)

  Obtains information about a particular mechanism of a token (in a slot)
- CK_RV C_InitToken (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen, CK_UTF8CHAR_PTR pLabel)

  Initializes a token (in a slot)
- CK_RV C_InitPIN (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)

  Initializes the normal user's PIN.
- CK_RV C_SetPIN (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pOldPin, CK_ULONG ulOld↩ Len, CK_UTF8CHAR_PTR pNewPin, CK_ULONG ulNewLen)

  Modifies the PIN of the current user.
- CK_RV C_OpenSession (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_NOTIFY notify, CK_SESSION_HANDLE_PTR phSession)

  Opens a connection between an application and a particular token or sets up an application callback for token insertion.
- CK_RV C_CloseSession (CK_SESSION_HANDLE hSession)

  Close the given session.
- CK_RV C_CloseAllSessions (CK_SLOT_ID slotID)

  Close all open sessions.
- CK_RV C_GetSessionInfo (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo)

  Retrieve information about the specified session.
- CK_RV C_GetOperationState (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pOperationState, CK_ULONG_PTR pulOperationStateLen)

  Obtains the cryptographic operations state of a session.
- CK_RV C_SetOperationState (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pOperationState, CK_ULONG ulOperationStateLen, CK_OBJECT_HANDLE hEncryptionKey, CK_OBJECT_HANDLE h↩ AuthenticationKey)

  Sets the cryptographic operations state of a session.
- CK_RV C_Login (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)

  Login on the token in the specified session.
- CK_RV C_Logout (CK_SESSION_HANDLE hSession)

  Log out of the token in the specified session.
- CK_RV C_CreateObject (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phObject)

  Create a new object on the token in the specified session using the given attribute template.
- CK_RV C_CopyObject (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phNewObject)

  Create a copy of the object with the specified handle.
- CK_RV C_DestroyObject (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Destroy the specified object.
- CK_RV C_GetObjectSize (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ULONG_PTR pulSize)

  Obtains the size of an object in bytes.
- CK_RV C_GetAttributeValue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Obtains an attribute value of an object.
- CK_RV C_SetAttributeValue (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Change or set the value of the specified attributes on the specified object.
- CK_RV C_FindObjectsInit (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)

  Initializes an object search in the specified session using the specified attribute template as search parameters.

- CK_RV C_FindObjects (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount)

    Continue the search for objects in the specified session.
- CK_RV C_FindObjectsFinal (CK_SESSION_HANDLE hSession)

    Finishes an object search operation (and cleans up)
- CK_RV C_EncryptInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)

    Initializes an encryption operation using the specified mechanism and session.
- CK_RV C_Encrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

    Perform a single operation encryption operation in the specified session.
- CK_RV C_EncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ul←DataLen, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

    Continues a multiple-part encryption operation.
- CK_RV C_EncryptFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG_PTR pulEncryptedDataLen)

    Finishes a multiple-part encryption operation.
- CK_RV C_DecryptInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hObject)

    Initialize decryption using the specified object.
- CK_RV C_Decrypt (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ul←EncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

    Perform a single operation decryption in the given session.
- CK_RV C_DecryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedData, CK_ULONG ulEncryptedDataLen, CK_BYTE_PTR pData, CK_ULONG_PTR pDataLen)

    Continues a multiple-part decryption operation.
- CK_RV C_DecryptFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG_PTR p←DataLen)

    Finishes a multiple-part decryption operation.
- CK_RV C_DigestInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism)

    Initializes a message-digesting operation using the specified mechanism in the specified session.
- CK_RV C_Digest (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

    Digest the specified data in a one-pass operation and return the resulting digest.
- CK_RV C_DigestUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

    Continues a multiple-part digesting operation.
- CK_RV C_DigestKey (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

    Update a running digest operation by digesting a secret key with the specified handle.
- CK_RV C_DigestFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pDigest, CK_ULONG_PTR pulDigestLen)

    Finishes a multiple-part digesting operation.
- CK_RV C_SignInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

    Initialize a signing operation using the specified key and mechanism.
- CK_RV C_Sign (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

    Sign the data in a single pass operation.
- CK_RV C_SignUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

    Continues a multiple-part signature operation.
- CK_RV C_SignFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

    Finishes a multiple-part signature operation.

- CK_RV C_SignRecoverInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a signature operation, where the data can be recovered from the signature.
- CK_RV C_SignRecover (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Signs single-part data, where the data can be recovered from the signature.
- CK_RV C_VerifyInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation using the specified key and mechanism.
- CK_RV C_Verify (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Verifies a signature on single-part data.
- CK_RV C_VerifyUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part verification operation.
- CK_RV C_VerifyFinal (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Finishes a multiple-part verification operation.
- CK_RV C_VerifyRecoverInit (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation where the data is recovered from the signature.
- CK_RV C_VerifyRecover (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen, CK_BYTE_PTR pData, CK_ULONG_PTR pulDataLen)

  Verifies a signature on single-part data, where the data is recovered from the signature.
- CK_RV C_DigestEncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pEncryptedPart, CK_ULONG_PTR pulEncryptedPartLen)

  Continues simultaneous multiple-part digesting and encryption operations.
- CK_RV C_DecryptDigestUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pDecryptedPart, CK_ULONG_PTR pulDecryptedPartLen)

  Continues simultaneous multiple-part decryption and digesting operations.
- CK_RV C_SignEncryptUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen, CK_BYTE_PTR pEncryptedPart, CK_ULONG_PTR pulEncryptedPartLen)

  Continues simultaneous multiple-part signature and encryption operations.
- CK_RV C_DecryptVerifyUpdate (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pEncryptedPart, CK_ULONG ulEncryptedPartLen, CK_BYTE_PTR pPart, CK_ULONG_PTR pulPartLen)

  Continues simultaneous multiple-part decryption and verification operations.
- CK_RV C_GenerateKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

  Generates a secret key using the specified mechanism.
- CK_RV C_GenerateKeyPair (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_ATTRIBUTE_PTR pPublicKeyTemplate, CK_ULONG ulPublicKeyAttributeCount, CK_ATTRIBUTE_PTR pPrivateKeyTemplate, CK_ULONG ulPrivateKeyAttributeCount, CK_OBJECT_HANDLE_PTR phPublicKey, CK_OBJECT_HANDLE_PTR phPrivateKey)

  Generates a public-key/private-key pair using the specified mechanism.
- CK_RV C_WrapKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hWrappingKey, CK_OBJECT_HANDLE hKey, CK_BYTE_PTR pWrappedKey, CK_ULONG_PTR pulWrappedKeyLen)

  Wraps (encrypts) the specified key using the specified wrapping key and mechanism.
- CK_RV C_UnwrapKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hUnwrappingKey, CK_BYTE_PTR pWrappedKey, CK_ULONG ulWrappedKeyLen, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

  Unwraps (decrypts) the specified key using the specified unwrapping key.
- CK_RV C_DeriveKey (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hBaseKey, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phKey)

Derive a key from the specified base key.

- CK_RV C_SeedRandom (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSeed, CK_ULONG ulSeedLen)

  Mixes in additional seed material to the random number generator.
- CK_RV C_GenerateRandom (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pRandomData, CK_ULONG ulRandomLen)

  Generate the specified amount of random data.
- CK_RV C_GetFunctionStatus (CK_SESSION_HANDLE hSession)

  Legacy function - see PKCS#11 v2.40.
- CK_RV C_CancelFunction (CK_SESSION_HANDLE hSession)

  Legacy function.
- CK_RV C_WaitForSlotEvent (CK_FLAGS flags, CK_SLOT_ID_PTR pSlot, CK_VOID_PTR pReserved)

  Wait for a slot event (token insertion, removal, etc) on the specified slot to occur.

10.159.1 Detailed Description

PKCS11 Basic library redirects based on the 2.40 specification `http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html`.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.160 pkcs11_mech.c File Reference

PKCS11 Library Mechanism Handling.

```
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_mech.h"
#include "pkcs11_slot.h"
#include "cryptoauthlib.h"
```

Data Structures

- struct _pcks11_mech_table_e

Macros

- #define PCKS11_MECH_ECC508_EC_CAPABILITY (CKF_EC_F_P | CKF_EC_NAMEDCURVE | CKF_EC_UNCOMPRESS)
- #define TABLE_SIZE(x) sizeof(x) / sizeof(x[0])

Typedefs

- typedef struct _pcks11_mech_table_e pcks11_mech_table_e
- typedef struct _pcks11_mech_table_e * pcks11_mech_table_ptr

Functions

- CK_RV pkcs11_mech_get_list (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount)
- CK_RV pkcs_mech_get_info (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo)

10.160.1 Detailed Description

PKCS11 Library Mechanism Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.161 pkcs11_mech.h File Reference

PKCS11 Library Mechanism Handling.

```
#include "cryptoki.h"
```

Functions

- CK_RV pkcs11_mech_get_list (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount)
- CK_RV pkcs_mech_get_info (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo)

10.161.1 Detailed Description

PKCS11 Library Mechanism Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.162 pkcs11_object.c File Reference

PKCS11 Library Object Handling Base.

```
#include "cryptoauthlib.h"
#include "cryptoki.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_session.h"
#include "pkcs11_util.h"
#include "pkcs11_object.h"
#include "pkcs11_os.h"
#include "pkcs11_find.h"
#include "pkcs11_key.h"
#include "pkcs11_cert.h"
```

Functions

- CK_RV pkcs11_object_alloc (pkcs11_object_ptr ∗ppObject)

  ∗∗
- CK_RV pkcs11_object_free (pkcs11_object_ptr pObject)
- CK_RV pkcs11_object_check (pkcs11_object_ptr ∗ppObject, CK_OBJECT_HANDLE hObject)
- CK_RV pkcs11_object_get_handle (pkcs11_object_ptr pObject, CK_OBJECT_HANDLE_PTR phObject)
- CK_RV pkcs11_object_get_name (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_class (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_destroyable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_size (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ULONG_PTR pulSize)
- CK_RV pkcs11_object_find (pkcs11_object_ptr ∗ppObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_object_create (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phObject)

  Create a new object on the token in the specified session using the given attribute template.
- CK_RV pkcs11_object_destroy (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Destroy the specified object.
- CK_RV pkcs11_object_deinit (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_object_load_handle_info (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_object_is_private (pkcs11_object_ptr pObject, CK_BBOOL ∗is_private)

  Checks the attributes of the underlying cryptographic asset to determine if it is a private key - this changes the way the associated public key is referenced.

Variables

- pkcs11_object_cache_t pkcs11_object_cache [PKCS11_MAX_OBJECTS_ALLOWED]
- const pkcs11_attrib_model pkcs11_object_monotonic_attributes [ ]
- const CK_ULONG pkcs11_object_monotonic_attributes_count = sizeof(pkcs11_object_monotonic_attributes) / sizeof(pkcs11_object_monotonic_attributes [0])

10.162.1 Detailed Description

PKCS11 Library Object Handling Base.

Copyright

10.163 pkcs11_object.h File Reference

PKCS11 Library Object Handling.

```
#include "cryptoauthlib.h"
#include "cryptoki.h"
#include "pkcs11_config.h"
#include "pkcs11_attrib.h"
```

Data Structures

- struct _pkcs11_object
- struct _pkcs11_object_cache_t

Macros

- #define PKCS11_OBJECT_FLAG_DESTROYABLE 0x01
- #define PKCS11_OBJECT_FLAG_MODIFIABLE 0x02
- #define PKCS11_OBJECT_FLAG_DYNAMIC 0x04
- #define PKCS11_OBJECT_FLAG_SENSITIVE 0x08
- #define PKCS11_OBJECT_FLAG_TA_TYPE 0x10
- #define PKCS11_OBJECT_FLAG_TRUST_TYPE 0x20

Typedefs

- typedef struct _pkcs11_object pkcs11_object
- typedef struct _pkcs11_object ∗ pkcs11_object_ptr
- typedef struct _pkcs11_object_cache_t pkcs11_object_cache_t

Functions

- CK_RV pkcs11_object_alloc (pkcs11_object_ptr ∗ppObject)

  ∗∗
- CK_RV pkcs11_object_free (pkcs11_object_ptr pObject)
- CK_RV pkcs11_object_check (pkcs11_object_ptr ∗ppObject, CK_OBJECT_HANDLE handle)
- CK_RV pkcs11_object_find (pkcs11_object_ptr ∗ppObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount)
- CK_RV pkcs11_object_is_private (pkcs11_object_ptr pObject, CK_BBOOL ∗is_private)

  Checks the attributes of the underlying cryptographic asset to determine if it is a private key - this changes the way the associated public key is referenced.
- CK_RV pkcs11_object_get_class (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_name (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_destroyable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_object_get_size (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ULONG_PTR pulSize)
- CK_RV pkcs11_object_get_handle (pkcs11_object_ptr pObject, CK_OBJECT_HANDLE_PTR phObject)
- CK_RV pkcs11_object_create (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount, CK_OBJECT_HANDLE_PTR phObject)

  Create a new object on the token in the specified session using the given attribute template.
- CK_RV pkcs11_object_destroy (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject)

  Destroy the specified object.
- CK_RV pkcs11_object_deinit (pkcs11_lib_ctx_ptr pContext)
- CK_RV pkcs11_object_load_handle_info (pkcs11_lib_ctx_ptr pContext)

Variables

- pkcs11_object_cache_t pkcs11_object_cache [ ]
- const pkcs11_attrib_model pkcs11_object_monotonic_attributes [ ]
- const CK_ULONG pkcs11_object_monotonic_attributes_count

10.163.1 Detailed Description

PKCS11 Library Object Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.163.2 Macro Definition Documentation

10.163.2.1 PKCS11_OBJECT_FLAG_DESTROYABLE

```
#define PKCS11_OBJECT_FLAG_DESTROYABLE 0x01
```

10.163.2.2 PKCS11_OBJECT_FLAG_DYNAMIC

```
#define PKCS11_OBJECT_FLAG_DYNAMIC 0x04
```

10.163.2.3 PKCS11_OBJECT_FLAG_MODIFIABLE

```
#define PKCS11_OBJECT_FLAG_MODIFIABLE 0x02
```

10.163.2.4 PKCS11_OBJECT_FLAG_SENSITIVE

```
#define PKCS11_OBJECT_FLAG_SENSITIVE 0x08
```

10.163.2.5 PKCS11_OBJECT_FLAG_TA_TYPE

```
#define PKCS11_OBJECT_FLAG_TA_TYPE 0x10
```

10.163.2.6 PKCS11_OBJECT_FLAG_TRUST_TYPE

```
#define PKCS11_OBJECT_FLAG_TRUST_TYPE 0x20
```

10.163.3 Typedef Documentation

10.163.3.1 pkcs11_object

```
typedef struct _pkcs11_object pkcs11_object
```

10.163.3.2 pkcs11_object_cache_t

```
typedef struct _pkcs11_object_cache_t pkcs11_object_cache_t
```

10.163.3.3 pkcs11_object_ptr

```
typedef struct _pkcs11_object * pkcs11_object_ptr
```

10.164 pkcs11_os.c File Reference

PKCS11 Library Operating System Abstraction Functions.

```
#include "pkcs11_os.h"
#include "pkcs11_util.h"
```

Functions

- CK_RV pkcs11_os_create_mutex (CK_VOID_PTR_PTR ppMutex)

  Application callback for creating a mutex object.
- CK_RV pkcs11_os_destroy_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_lock_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_unlock_mutex (CK_VOID_PTR pMutex)

10.164.1 Detailed Description

PKCS11 Library Operating System Abstraction Functions.

Copyright

10.165 pkcs11_os.h File Reference

PKCS11 Library Operating System Abstraction.

```
#include "cryptoki.h"
#include "cryptoauthlib.h"
```

Macros

- #define pkcs11_os_malloc hal_malloc
- #define pkcs11_os_free hal_free

Functions

- CK_RV pkcs11_os_create_mutex (CK_VOID_PTR_PTR ppMutex)

  Application callback for creating a mutex object.
- CK_RV pkcs11_os_destroy_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_lock_mutex (CK_VOID_PTR pMutex)
- CK_RV pkcs11_os_unlock_mutex (CK_VOID_PTR pMutex)

10.165.1 Detailed Description

PKCS11 Library Operating System Abstraction.

Copyright

10.165.2 Macro Definition Documentation

10.165.2.1 pkcs11_os_free

```
#define pkcs11_os_free hal_free
```

10.165.2.2 pkcs11_os_malloc

```
#define pkcs11_os_malloc hal_malloc
```

10.166 pkcs11_session.c File Reference

PKCS11 Library Session Handling.

```
#include "cryptoauthlib.h"
#include "crypto/atca_crypto_sw_rand.h"
#include "host/atca_host.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_session.h"
#include "pkcs11_token.h"
#include "pkcs11_init.h"
#include "pkcs11_slot.h"
#include "pkcs11_object.h"
#include "pkcs11_os.h"
#include "pkcs11_util.h"
```

Functions

- pkcs11_session_ctx_ptr pkcs11_get_session_context (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_session_check (pkcs11_session_ctx_ptr *pSession, CK_SESSION_HANDLE hSession)

  Check if the session is initialized properly.
- CK_RV pkcs11_session_open (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_↩ NOTIFY notify, CK_SESSION_HANDLE_PTR phSession)
- CK_RV pkcs11_session_close (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_session_closeall (CK_SLOT_ID slotID)

  Close all sessions for a given slot - not actually all open sessions.
- CK_RV pkcs11_session_get_info (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo)

  Obtains information about a particular session.
- CK_RV pkcs11_session_login (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)
- CK_RV pkcs11_session_logout (CK_SESSION_HANDLE hSession)

10.166.1 Detailed Description

PKCS11 Library Session Handling.

Copyright

10.167 pkcs11_session.h File Reference

PKCS11 Library Session Management & Context.

```
#include "cryptoki.h"
#include "pkcs11_config.h"
```

Data Structures

- union _pkcs11_session_mech_ctx
- struct _pkcs11_session_ctx

Typedefs

- typedef union _pkcs11_session_mech_ctx pkcs11_session_mech_ctx
- typedef union _pkcs11_session_mech_ctx * pkcs11_session_mech_ctx_ptr
- typedef struct _pkcs11_session_ctx pkcs11_session_ctx
- typedef struct _pkcs11_session_ctx * pkcs11_session_ctx_ptr

Functions

- CK_RV pkcs11_session_check (pkcs11_session_ctx_ptr *pSession, CK_SESSION_HANDLE hSession)

  Check if the session is initialized properly.
- CK_RV pkcs11_session_get_info (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo)

  Obtains information about a particular session.
- CK_RV pkcs11_session_open (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_↩ NOTIFY notify, CK_SESSION_HANDLE_PTR phSession)
- CK_RV pkcs11_session_close (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_session_closeall (CK_SLOT_ID slotID)

  Close all sessions for a given slot - not actually all open sessions.
- CK_RV pkcs11_session_login (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen)
- CK_RV pkcs11_session_logout (CK_SESSION_HANDLE hSession)
- CK_RV pkcs11_session_authorize (pkcs11_session_ctx_ptr pSession, CK_VOID_PTR pObject)

10.167.1 Detailed Description

PKCS11 Library Session Management & Context.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.167.2 Typedef Documentation

10.167.2.1 pkcs11_session_ctx

```
typedef struct _pkcs11_session_ctx pkcs11_session_ctx
```

Session Context

10.167.2.2 pkcs11_session_ctx_ptr

```
typedef struct _pkcs11_session_ctx * pkcs11_session_ctx_ptr
```

10.167.2.3 pkcs11_session_mech_ctx

```
typedef union _pkcs11_session_mech_ctx pkcs11_session_mech_ctx
```

10.167.2.4 pkcs11_session_mech_ctx_ptr

```
typedef union _pkcs11_session_mech_ctx * pkcs11_session_mech_ctx_ptr
```

10.167.3 Function Documentation

10.167.3.1 pkcs11_session_authorize()

```
CK_RV pkcs11_session_authorize (
            pkcs11_session_ctx_ptr pSession,
            CK_VOID_PTR pObject )
```

10.168 pkcs11_signature.c File Reference

PKCS11 Library Sign/Verify Handling.

```
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_signature.h"
#include "pkcs11_object.h"
#include "pkcs11_session.h"
#include "pkcs11_util.h"
#include "cryptoauthlib.h"
#include "atcacert/atcacert_der.h"
```

Functions

- CK_RV pkcs11_signature_sign_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p↩ Mechanism, CK_OBJECT_HANDLE hKey)

  Initialize a signing operation using the specified key and mechanism.
- CK_RV pkcs11_signature_sign (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Sign the data in a single pass operation.
- CK_RV pkcs11_signature_sign_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part signature operation.
- CK_RV pkcs11_signature_sign_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Finishes a multiple-part signature operation.
- CK_RV pkcs11_signature_verify_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p↩ Mechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation using the specified key and mechanism.
- CK_RV pkcs11_signature_verify (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Verifies a signature on single-part data.
- CK_RV pkcs11_signature_verify_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part verification operation.
- CK_RV pkcs11_signature_verify_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Finishes a multiple-part verification operation.

10.168.1 Detailed Description

PKCS11 Library Sign/Verify Handling.

Copyright

10.169 pkcs11_signature.h File Reference

PKCS11 Library Sign/Verify Handling.

```
#include "cryptoki.h"
```

Functions

- CK_RV pkcs11_signature_sign_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p↩ Mechanism, CK_OBJECT_HANDLE hKey)

  Initialize a signing operation using the specified key and mechanism.
- CK_RV pkcs11_signature_sign (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Sign the data in a single pass operation.
- CK_RV pkcs11_signature_sign_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part signature operation.
- CK_RV pkcs11_signature_sign_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen)

  Finishes a multiple-part signature operation.
- CK_RV pkcs11_signature_verify_init (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR p↩ Mechanism, CK_OBJECT_HANDLE hKey)

  Initializes a verification operation using the specified key and mechanism.
- CK_RV pkcs11_signature_verify (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Verifies a signature on single-part data.
- CK_RV pkcs11_signature_verify_continue (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pPart, CK_ULONG ulPartLen)

  Continues a multiple-part verification operation.
- CK_RV pkcs11_signature_verify_finish (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pSignature, CK_ULONG ulSignatureLen)

  Finishes a multiple-part verification operation.

10.169.1 Detailed Description

PKCS11 Library Sign/Verify Handling.

Copyright

10.170 pkcs11_slot.c File Reference

PKCS11 Library Slot Handling.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_init.h"
#include "pkcs11_slot.h"
#include "pkcs11_info.h"
#include "pkcs11_util.h"
#include "pkcs11_object.h"
#include "pkcs11_os.h"
#include <stdio.h>
```

Functions

- pkcs11_slot_ctx_ptr pkcs11_slot_get_context (pkcs11_lib_ctx_ptr lib_ctx, CK_SLOT_ID slotID)
    Retrieve the current slot context.
- CK_VOID_PTR pkcs11_slot_initslots (CK_ULONG pulCount)
- CK_RV pkcs11_slot_config (CK_SLOT_ID slotID)
- CK_RV pkcs11_slot_init (CK_SLOT_ID slotID)
- CK_RV pkcs11_slot_get_list (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pulCount)
- CK_RV pkcs11_slot_get_info (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo)
    Obtains information about a particular slot.

10.170.1 Detailed Description

PKCS11 Library Slot Handling.

The nomenclature here can lead to some confusion - the pkcs11 slot is not the same as a device slot. So for example each slot defined here is a specific device (most systems would have only one). The "slots" as defined by the device specification would be enumerated seperately as related to specific supported mechanisms as cryptographic "objects".

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.171 pkcs11_slot.h File Reference

PKCS11 Library Slot Handling & Context.

```
#include "pkcs11_init.h"
#include "cryptoauthlib.h"
```

Data Structures

- struct _pkcs11_slot_ctx

Typedefs

- typedef struct _pkcs11_slot_ctx pkcs11_slot_ctx
- typedef struct _pkcs11_slot_ctx * pkcs11_slot_ctx_ptr

Functions

- CK_RV pkcs11_slot_init (CK_SLOT_ID slotID)
- CK_RV pkcs11_slot_config (CK_SLOT_ID slotID)
- CK_VOID_PTR pkcs11_slot_initslots (CK_ULONG pulCount)
- pkcs11_slot_ctx_ptr pkcs11_slot_get_context (pkcs11_lib_ctx_ptr lib_ctx, CK_SLOT_ID slotID)
    Retrieve the current slot context.
- CK_RV pkcs11_slot_get_list (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pulCount)
- CK_RV pkcs11_slot_get_info (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo)
    Obtains information about a particular slot.

10.171.1 Detailed Description

PKCS11 Library Slot Handling & Context.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.171.2 Typedef Documentation

10.171.2.1 pkcs11_slot_ctx

```
typedef struct _pkcs11_slot_ctx pkcs11_slot_ctx
```

Slot Context

10.171.2.2 pkcs11_slot_ctx_ptr

```
typedef struct _pkcs11_slot_ctx * pkcs11_slot_ctx_ptr
```

10.172 pkcs11_token.c File Reference

PKCS11 Library Token Handling.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11_debug.h"
#include "pkcs11_token.h"
#include "pkcs11_slot.h"
#include "pkcs11_info.h"
#include "pkcs11_util.h"
#include "pkcs11_object.h"
#include "pkcs11_key.h"
#include "pkcs11_cert.h"
#include "pkcs11_session.h"
```

Macros

- #define ATCA_SERIAL_NUM_SIZE (9)

Functions

- CK_RV pkcs11_token_init (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen, CK_UTF8CHAR_PTR pLabel)
- CK_RV pkcs11_token_get_access_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_writable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_storage (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_info (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo)

  Obtains information about a particular token.
- CK_RV pkcs11_token_random (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pRandomData, CK_ULONG ulRandomLen)

  Generate the specified amount of random data.
- CK_RV pkcs11_token_convert_pin_to_key (const CK_UTF8CHAR_PTR pPin, const CK_ULONG ulPinLen, const CK_UTF8CHAR_PTR pSalt, const CK_ULONG ulSaltLen, CK_BYTE_PTR pKey, CK_ULONG ulKey↵ Len)
- CK_RV pkcs11_token_set_pin (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pOldPin, CK_ULONG ulOldLen, CK_UTF8CHAR_PTR pNewPin, CK_ULONG ulNewLen)

10.172.1 Detailed Description

PKCS11 Library Token Handling.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.172.2 Macro Definition Documentation

10.172.2.1 ATCA_SERIAL_NUM_SIZE

```
#define ATCA_SERIAL_NUM_SIZE (9)
```

10.173 pkcs11_token.h File Reference

PKCS11 Library Token Management & Context.

```
#include "pkcs11_init.h"
```

Functions

- CK_RV pkcs11_token_init (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen, CK_UTF8CHAR_PTR pLabel)
- CK_RV pkcs11_token_get_access_type (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_writable (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_storage (CK_VOID_PTR pObject, CK_ATTRIBUTE_PTR pAttribute)
- CK_RV pkcs11_token_get_info (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo)

    Obtains information about a particular token.
- CK_RV pkcs11_token_convert_pin_to_key (const CK_UTF8CHAR_PTR pPin, const CK_ULONG ulPinLen, const CK_UTF8CHAR_PTR pSalt, const CK_ULONG ulSaltLen, CK_BYTE_PTR pKey, CK_ULONG ulKey←Len)
- CK_RV pkcs11_token_random (CK_SESSION_HANDLE hSession, CK_BYTE_PTR pRandomData, CK_ULONG ulRandomLen)

    Generate the specified amount of random data.
- CK_RV pkcs11_token_set_pin (CK_SESSION_HANDLE hSession, CK_UTF8CHAR_PTR pOldPin, CK_ULONG ulOldLen, CK_UTF8CHAR_PTR pNewPin, CK_ULONG ulNewLen)

10.173.1 Detailed Description

PKCS11 Library Token Management & Context.

Copyright

10.174 pkcs11_util.c File Reference

PKCS11 Library Utility Functions.

```
#include "pkcs11_util.h"
```

Functions

- void pkcs11_util_escape_string (CK_UTF8CHAR_PTR buf, CK_ULONG buf_len)
- CK_RV pkcs11_util_convert_rv (ATCA_STATUS status)
- int pkcs11_util_memset (void ∗dest, size_t destsz, int ch, size_t count)

10.174.1 Detailed Description

PKCS11 Library Utility Functions.

Copyright

10.175 pkcs11_util.h File Reference

PKCS11 Library Utilities.

```
#include "pkcs11_config.h"
#include "cryptoki.h"
#include "cryptoauthlib.h"
```

Macros

- #define PKCS11_UTIL_ARRAY_SIZE(x) sizeof(x) / sizeof(x[0])

Functions

- void pkcs11_util_escape_string (CK_UTF8CHAR_PTR buf, CK_ULONG buf_len)
- CK_RV pkcs11_util_convert_rv (ATCA_STATUS status)
- int pkcs11_util_memset (void ∗dest, size_t destsz, int ch, size_t count)

10.175.1 Detailed Description

PKCS11 Library Utilities.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.175.2 Macro Definition Documentation

10.175.2.1 PKCS11_UTIL_ARRAY_SIZE

```
#define PKCS11_UTIL_ARRAY_SIZE(
            x ) sizeof(x) / sizeof(x[0])
```

10.176 pkcs11f.h File Reference

10.177 pkcs11t.h File Reference

Data Structures

- struct CK_VERSION
- struct CK_INFO
- struct CK_SLOT_INFO
- struct CK_TOKEN_INFO
- struct CK_SESSION_INFO
- struct CK_ATTRIBUTE
- struct CK_DATE
- struct CK_MECHANISM
- struct CK_MECHANISM_INFO
- struct CK_C_INITIALIZE_ARGS
- struct CK_RSA_PKCS_OAEP_PARAMS
- struct CK_RSA_PKCS_PSS_PARAMS
- struct CK_ECDH1_DERIVE_PARAMS
- struct CK_ECDH2_DERIVE_PARAMS
- struct CK_ECMQV_DERIVE_PARAMS
- struct CK_X9_42_DH1_DERIVE_PARAMS
- struct CK_X9_42_DH2_DERIVE_PARAMS
- struct CK_X9_42_MQV_DERIVE_PARAMS
- struct CK_KEA_DERIVE_PARAMS
- struct CK_RC2_CBC_PARAMS
- struct CK_RC2_MAC_GENERAL_PARAMS
- struct CK_RC5_PARAMS
- struct CK_RC5_CBC_PARAMS
- struct CK_RC5_MAC_GENERAL_PARAMS
- struct CK_DES_CBC_ENCRYPT_DATA_PARAMS
- struct CK_AES_CBC_ENCRYPT_DATA_PARAMS
- struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS
- struct CK_SKIPJACK_RELAYX_PARAMS
- struct CK_PBE_PARAMS
- struct CK_KEY_WRAP_SET_OAEP_PARAMS
- struct CK_SSL3_RANDOM_DATA
- struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS
- struct CK_SSL3_KEY_MAT_OUT
- struct CK_SSL3_KEY_MAT_PARAMS
- struct CK_TLS_PRF_PARAMS
- struct CK_WTLS_RANDOM_DATA
- struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS
- struct CK_WTLS_PRF_PARAMS
- struct CK_WTLS_KEY_MAT_OUT
- struct CK_WTLS_KEY_MAT_PARAMS
- struct CK_CMS_SIG_PARAMS
- struct CK_KEY_DERIVATION_STRING_DATA
- struct CK_PKCS5_PBKD2_PARAMS
- struct CK_PKCS5_PBKD2_PARAMS2
- struct CK_OTP_PARAM
- struct CK_OTP_PARAMS
- struct CK_OTP_SIGNATURE_INFO

- struct CK_KIP_PARAMS
- struct CK_AES_CTR_PARAMS
- struct CK_GCM_PARAMS
- struct CK_CCM_PARAMS
- struct CK_AES_GCM_PARAMS
- struct CK_AES_CCM_PARAMS
- struct CK_CAMELLIA_CTR_PARAMS
- struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS
- struct CK_ARIA_CBC_ENCRYPT_DATA_PARAMS
- struct CK_DSA_PARAMETER_GEN_PARAM
- struct CK_ECDH_AES_KEY_WRAP_PARAMS
- struct CK_RSA_AES_KEY_WRAP_PARAMS
- struct CK_TLS12_MASTER_KEY_DERIVE_PARAMS
- struct CK_TLS12_KEY_MAT_PARAMS
- struct CK_TLS_KDF_PARAMS
- struct CK_TLS_MAC_PARAMS
- struct CK_GOSTR3410_DERIVE_PARAMS
- struct CK_GOSTR3410_KEY_WRAP_PARAMS
- struct CK_SEED_CBC_ENCRYPT_DATA_PARAMS

Macros

- #define CRYPTOKI_VERSION_MAJOR 2
- #define CRYPTOKI_VERSION_MINOR 40
- #define CRYPTOKI_VERSION_AMENDMENT 0
- #define CK_TRUE 1
- #define CK_FALSE 0
- #define FALSE CK_FALSE
- #define TRUE CK_TRUE
- #define CK_UNAVAILABLE_INFORMATION (∼0UL)
- #define CK_EFFECTIVELY_INFINITE 0UL
- #define CK_INVALID_HANDLE 0UL
- #define CKN_SURRENDER 0UL
- #define CKN_OTP_CHANGED 1UL
- #define CKF_TOKEN_PRESENT 0x00000001UL /∗ a token is there ∗/
- #define CKF_REMOVABLE_DEVICE 0x00000002UL /∗ removable devices∗/
- #define CKF_HW_SLOT 0x00000004UL /∗ hardware slot ∗/
- #define CKF_RNG 0x00000001UL /∗ has random # generator ∗/
- #define CKF_WRITE_PROTECTED 0x00000002UL /∗ token is write-protected ∗/
- #define CKF_LOGIN_REQUIRED 0x00000004UL /∗ user must login ∗/
- #define CKF_USER_PIN_INITIALIZED 0x00000008UL /∗ normal user's PIN is set ∗/
- #define CKF_RESTORE_KEY_NOT_NEEDED 0x00000020UL
- #define CKF_CLOCK_ON_TOKEN 0x00000040UL
- #define CKF_PROTECTED_AUTHENTICATION_PATH 0x00000100UL
- #define CKF_DUAL_CRYPTO_OPERATIONS 0x00000200UL
- #define CKF_TOKEN_INITIALIZED 0x00000400UL
- #define CKF_SECONDARY_AUTHENTICATION 0x00000800UL
- #define CKF_USER_PIN_COUNT_LOW 0x00010000UL
- #define CKF_USER_PIN_FINAL_TRY 0x00020000UL
- #define CKF_USER_PIN_LOCKED 0x00040000UL
- #define CKF_USER_PIN_TO_BE_CHANGED 0x00080000UL
- #define CKF_SO_PIN_COUNT_LOW 0x00100000UL
- #define CKF_SO_PIN_FINAL_TRY 0x00200000UL
- #define CKF_SO_PIN_LOCKED 0x00400000UL

- #define CKF_SO_PIN_TO_BE_CHANGED 0x00800000UL
- #define CKF_ERROR_STATE 0x01000000UL
- #define CKU_SO 0UL
- #define CKU_USER 1UL
- #define CKU_CONTEXT_SPECIFIC 2UL
- #define CKS_RO_PUBLIC_SESSION 0UL
- #define CKS_RO_USER_FUNCTIONS 1UL
- #define CKS_RW_PUBLIC_SESSION 2UL
- #define CKS_RW_USER_FUNCTIONS 3UL
- #define CKS_RW_SO_FUNCTIONS 4UL
- #define CKF_RW_SESSION 0x00000002UL /∗ session is r/w ∗/
- #define CKF_SERIAL_SESSION 0x00000004UL /∗ no parallel ∗/
- #define CKO_DATA 0x00000000UL
- #define CKO_CERTIFICATE 0x00000001UL
- #define CKO_PUBLIC_KEY 0x00000002UL
- #define CKO_PRIVATE_KEY 0x00000003UL
- #define CKO_SECRET_KEY 0x00000004UL
- #define CKO_HW_FEATURE 0x00000005UL
- #define CKO_DOMAIN_PARAMETERS 0x00000006UL
- #define CKO_MECHANISM 0x00000007UL
- #define CKO_OTP_KEY 0x00000008UL
- #define CKO_VENDOR_DEFINED 0x80000000UL
- #define CKH_MONOTONIC_COUNTER 0x00000001UL
- #define CKH_CLOCK 0x00000002UL
- #define CKH_USER_INTERFACE 0x00000003UL
- #define CKH_VENDOR_DEFINED 0x80000000UL
- #define CKK_RSA 0x00000000UL
- #define CKK_DSA 0x00000001UL
- #define CKK_DH 0x00000002UL
- #define CKK_ECDSA 0x00000003UL /∗ Deprecated ∗/
- #define CKK_EC 0x00000003UL
- #define CKK_X9_42_DH 0x00000004UL
- #define CKK_KEA 0x00000005UL
- #define CKK_GENERIC_SECRET 0x00000010UL
- #define CKK_RC2 0x00000011UL
- #define CKK_RC4 0x00000012UL
- #define CKK_DES 0x00000013UL
- #define CKK_DES2 0x00000014UL
- #define CKK_DES3 0x00000015UL
- #define CKK_CAST 0x00000016UL
- #define CKK_CAST3 0x00000017UL
- #define CKK_CAST5 0x00000018UL /∗ Deprecated ∗/
- #define CKK_CAST128 0x00000018UL
- #define CKK_RC5 0x00000019UL
- #define CKK_IDEA 0x0000001AUL
- #define CKK_SKIPJACK 0x0000001BUL
- #define CKK_BATON 0x0000001CUL
- #define CKK_JUNIPER 0x0000001DUL
- #define CKK_CDMF 0x0000001EUL
- #define CKK_AES 0x0000001FUL
- #define CKK_BLOWFISH 0x00000020UL
- #define CKK_TWOFISH 0x00000021UL
- #define CKK_SECURID 0x00000022UL
- #define CKK_HOTP 0x00000023UL
- #define CKK_ACTI 0x00000024UL

- #define CKK_CAMELLIA 0x00000025UL
- #define CKK_ARIA 0x00000026UL
- #define CKK_MD5_HMAC 0x00000027UL
- #define CKK_SHA_1_HMAC 0x00000028UL
- #define CKK_RIPEMD128_HMAC 0x00000029UL
- #define CKK_RIPEMD160_HMAC 0x0000002AUL
- #define CKK_SHA256_HMAC 0x0000002BUL
- #define CKK_SHA384_HMAC 0x0000002CUL
- #define CKK_SHA512_HMAC 0x0000002DUL
- #define CKK_SHA224_HMAC 0x0000002EUL
- #define CKK_SEED 0x0000002FUL
- #define CKK_GOSTR3410 0x00000030UL
- #define CKK_GOSTR3411 0x00000031UL
- #define CKK_GOST28147 0x00000032UL
- #define CKK_VENDOR_DEFINED 0x80000000UL
- #define CK_CERTIFICATE_CATEGORY_UNSPECIFIED 0UL
- #define CK_CERTIFICATE_CATEGORY_TOKEN_USER 1UL
- #define CK_CERTIFICATE_CATEGORY_AUTHORITY 2UL
- #define CK_CERTIFICATE_CATEGORY_OTHER_ENTITY 3UL
- #define CK_SECURITY_DOMAIN_UNSPECIFIED 0UL
- #define CK_SECURITY_DOMAIN_MANUFACTURER 1UL
- #define CK_SECURITY_DOMAIN_OPERATOR 2UL
- #define CK_SECURITY_DOMAIN_THIRD_PARTY 3UL
- #define CKC_X_509 0x00000000UL
- #define CKC_X_509_ATTR_CERT 0x00000001UL
- #define CKC_WTLS 0x00000002UL
- #define CKC_VENDOR_DEFINED 0x80000000UL
- #define CKC_OPENPGP (CKC_VENDOR_DEFINED | 0x00504750)
- #define CKF_ARRAY_ATTRIBUTE 0x40000000UL
- #define CK_OTP_FORMAT_DECIMAL 0UL
- #define CK_OTP_FORMAT_HEXADECIMAL 1UL
- #define CK_OTP_FORMAT_ALPHANUMERIC 2UL
- #define CK_OTP_FORMAT_BINARY 3UL
- #define CK_OTP_PARAM_IGNORED 0UL
- #define CK_OTP_PARAM_OPTIONAL 1UL
- #define CK_OTP_PARAM_MANDATORY 2UL
- #define CKA_CLASS 0x00000000UL
- #define CKA_TOKEN 0x00000001UL
- #define CKA_PRIVATE 0x00000002UL
- #define CKA_LABEL 0x00000003UL
- #define CKA_APPLICATION 0x00000010UL
- #define CKA_VALUE 0x00000011UL
- #define CKA_OBJECT_ID 0x00000012UL
- #define CKA_CERTIFICATE_TYPE 0x00000080UL
- #define CKA_ISSUER 0x00000081UL
- #define CKA_SERIAL_NUMBER 0x00000082UL
- #define CKA_AC_ISSUER 0x00000083UL
- #define CKA_OWNER 0x00000084UL
- #define CKA_ATTR_TYPES 0x00000085UL
- #define CKA_TRUSTED 0x00000086UL
- #define CKA_CERTIFICATE_CATEGORY 0x00000087UL
- #define CKA_JAVA_MIDP_SECURITY_DOMAIN 0x00000088UL
- #define CKA_URL 0x00000089UL
- #define CKA_HASH_OF_SUBJECT_PUBLIC_KEY 0x0000008AUL
- #define CKA_HASH_OF_ISSUER_PUBLIC_KEY 0x0000008BUL

- #define CKA_NAME_HASH_ALGORITHM 0x0000008CUL
- #define CKA_CHECK_VALUE 0x00000090UL
- #define CKA_KEY_TYPE 0x00000100UL
- #define CKA_SUBJECT 0x00000101UL
- #define CKA_ID 0x00000102UL
- #define CKA_SENSITIVE 0x00000103UL
- #define CKA_ENCRYPT 0x00000104UL
- #define CKA_DECRYPT 0x00000105UL
- #define CKA_WRAP 0x00000106UL
- #define CKA_UNWRAP 0x00000107UL
- #define CKA_SIGN 0x00000108UL
- #define CKA_SIGN_RECOVER 0x00000109UL
- #define CKA_VERIFY 0x0000010AUL
- #define CKA_VERIFY_RECOVER 0x0000010BUL
- #define CKA_DERIVE 0x0000010CUL
- #define CKA_START_DATE 0x00000110UL
- #define CKA_END_DATE 0x00000111UL
- #define CKA_MODULUS 0x00000120UL
- #define CKA_MODULUS_BITS 0x00000121UL
- #define CKA_PUBLIC_EXPONENT 0x00000122UL
- #define CKA_PRIVATE_EXPONENT 0x00000123UL
- #define CKA_PRIME_1 0x00000124UL
- #define CKA_PRIME_2 0x00000125UL
- #define CKA_EXPONENT_1 0x00000126UL
- #define CKA_EXPONENT_2 0x00000127UL
- #define CKA_COEFFICIENT 0x00000128UL
- #define CKA_PUBLIC_KEY_INFO 0x00000129UL
- #define CKA_PRIME 0x00000130UL
- #define CKA_SUBPRIME 0x00000131UL
- #define CKA_BASE 0x00000132UL
- #define CKA_PRIME_BITS 0x00000133UL
- #define CKA_SUBPRIME_BITS 0x00000134UL
- #define CKA_SUB_PRIME_BITS CKA_SUBPRIME_BITS
- #define CKA_VALUE_BITS 0x00000160UL
- #define CKA_VALUE_LEN 0x00000161UL
- #define CKA_EXTRACTABLE 0x00000162UL
- #define CKA_LOCAL 0x00000163UL
- #define CKA_NEVER_EXTRACTABLE 0x00000164UL
- #define CKA_ALWAYS_SENSITIVE 0x00000165UL
- #define CKA_KEY_GEN_MECHANISM 0x00000166UL
- #define CKA_MODIFIABLE 0x00000170UL
- #define CKA_COPYABLE 0x00000171UL
- #define CKA_DESTROYABLE 0x00000172UL
- #define CKA_ECDSA_PARAMS 0x00000180UL /∗ Deprecated ∗/
- #define CKA_EC_PARAMS 0x00000180UL
- #define CKA_EC_POINT 0x00000181UL
- #define CKA_SECONDARY_AUTH 0x00000200UL /∗ Deprecated ∗/
- #define CKA_AUTH_PIN_FLAGS 0x00000201UL /∗ Deprecated ∗/
- #define CKA_ALWAYS_AUTHENTICATE 0x00000202UL
- #define CKA_WRAP_WITH_TRUSTED 0x00000210UL
- #define CKA_WRAP_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000211UL)
- #define CKA_UNWRAP_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000212UL)
- #define CKA_DERIVE_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000213UL)
- #define CKA_OTP_FORMAT 0x00000220UL
- #define CKA_OTP_LENGTH 0x00000221UL

- #define CKA_OTP_TIME_INTERVAL 0x00000222UL
- #define CKA_OTP_USER_FRIENDLY_MODE 0x00000223UL
- #define CKA_OTP_CHALLENGE_REQUIREMENT 0x00000224UL
- #define CKA_OTP_TIME_REQUIREMENT 0x00000225UL
- #define CKA_OTP_COUNTER_REQUIREMENT 0x00000226UL
- #define CKA_OTP_PIN_REQUIREMENT 0x00000227UL
- #define CKA_OTP_COUNTER 0x0000022EUL
- #define CKA_OTP_TIME 0x0000022FUL
- #define CKA_OTP_USER_IDENTIFIER 0x0000022AUL
- #define CKA_OTP_SERVICE_IDENTIFIER 0x0000022BUL
- #define CKA_OTP_SERVICE_LOGO 0x0000022CUL
- #define CKA_OTP_SERVICE_LOGO_TYPE 0x0000022DUL
- #define CKA_GOSTR3410_PARAMS 0x00000250UL
- #define CKA_GOSTR3411_PARAMS 0x00000251UL
- #define CKA_GOST28147_PARAMS 0x00000252UL
- #define CKA_HW_FEATURE_TYPE 0x00000300UL
- #define CKA_RESET_ON_INIT 0x00000301UL
- #define CKA_HAS_RESET 0x00000302UL
- #define CKA_PIXEL_X 0x00000400UL
- #define CKA_PIXEL_Y 0x00000401UL
- #define CKA_RESOLUTION 0x00000402UL
- #define CKA_CHAR_ROWS 0x00000403UL
- #define CKA_CHAR_COLUMNS 0x00000404UL
- #define CKA_COLOR 0x00000405UL
- #define CKA_BITS_PER_PIXEL 0x00000406UL
- #define CKA_CHAR_SETS 0x00000480UL
- #define CKA_ENCODING_METHODS 0x00000481UL
- #define CKA_MIME_TYPES 0x00000482UL
- #define CKA_MECHANISM_TYPE 0x00000500UL
- #define CKA_REQUIRED_CMS_ATTRIBUTES 0x00000501UL
- #define CKA_DEFAULT_CMS_ATTRIBUTES 0x00000502UL
- #define CKA_SUPPORTED_CMS_ATTRIBUTES 0x00000503UL
- #define CKA_ALLOWED_MECHANISMS (CKF_ARRAY_ATTRIBUTE | 0x00000600UL)
- #define CKA_VENDOR_DEFINED 0x80000000UL
- #define CKM_RSA_PKCS_KEY_PAIR_GEN 0x00000000UL
- #define CKM_RSA_PKCS 0x00000001UL
- #define CKM_RSA_9796 0x00000002UL
- #define CKM_RSA_X_509 0x00000003UL
- #define CKM_MD2_RSA_PKCS 0x00000004UL
- #define CKM_MD5_RSA_PKCS 0x00000005UL
- #define CKM_SHA1_RSA_PKCS 0x00000006UL
- #define CKM_RIPEMD128_RSA_PKCS 0x00000007UL
- #define CKM_RIPEMD160_RSA_PKCS 0x00000008UL
- #define CKM_RSA_PKCS_OAEP 0x00000009UL
- #define CKM_RSA_X9_31_KEY_PAIR_GEN 0x0000000AUL
- #define CKM_RSA_X9_31 0x0000000BUL
- #define CKM_SHA1_RSA_X9_31 0x0000000CUL
- #define CKM_RSA_PKCS_PSS 0x0000000DUL
- #define CKM_SHA1_RSA_PKCS_PSS 0x0000000EUL
- #define CKM_DSA_KEY_PAIR_GEN 0x00000010UL
- #define CKM_DSA 0x00000011UL
- #define CKM_DSA_SHA1 0x00000012UL
- #define CKM_DSA_SHA224 0x00000013UL
- #define CKM_DSA_SHA256 0x00000014UL
- #define CKM_DSA_SHA384 0x00000015UL

- #define CKM_DSA_SHA512 0x00000016UL
- #define CKM_DH_PKCS_KEY_PAIR_GEN 0x00000020UL
- #define CKM_DH_PKCS_DERIVE 0x00000021UL
- #define CKM_X9_42_DH_KEY_PAIR_GEN 0x00000030UL
- #define CKM_X9_42_DH_DERIVE 0x00000031UL
- #define CKM_X9_42_DH_HYBRID_DERIVE 0x00000032UL
- #define CKM_X9_42_MQV_DERIVE 0x00000033UL
- #define CKM_SHA256_RSA_PKCS 0x00000040UL
- #define CKM_SHA384_RSA_PKCS 0x00000041UL
- #define CKM_SHA512_RSA_PKCS 0x00000042UL
- #define CKM_SHA256_RSA_PKCS_PSS 0x00000043UL
- #define CKM_SHA384_RSA_PKCS_PSS 0x00000044UL
- #define CKM_SHA512_RSA_PKCS_PSS 0x00000045UL
- #define CKM_SHA224_RSA_PKCS 0x00000046UL
- #define CKM_SHA224_RSA_PKCS_PSS 0x00000047UL
- #define CKM_SHA512_224 0x00000048UL
- #define CKM_SHA512_224_HMAC 0x00000049UL
- #define CKM_SHA512_224_HMAC_GENERAL 0x0000004AUL
- #define CKM_SHA512_224_KEY_DERIVATION 0x0000004BUL
- #define CKM_SHA512_256 0x0000004CUL
- #define CKM_SHA512_256_HMAC 0x0000004DUL
- #define CKM_SHA512_256_HMAC_GENERAL 0x0000004EUL
- #define CKM_SHA512_256_KEY_DERIVATION 0x0000004FUL
- #define CKM_SHA512_T 0x00000050UL
- #define CKM_SHA512_T_HMAC 0x00000051UL
- #define CKM_SHA512_T_HMAC_GENERAL 0x00000052UL
- #define CKM_SHA512_T_KEY_DERIVATION 0x00000053UL
- #define CKM_RC2_KEY_GEN 0x00000100UL
- #define CKM_RC2_ECB 0x00000101UL
- #define CKM_RC2_CBC 0x00000102UL
- #define CKM_RC2_MAC 0x00000103UL
- #define CKM_RC2_MAC_GENERAL 0x00000104UL
- #define CKM_RC2_CBC_PAD 0x00000105UL
- #define CKM_RC4_KEY_GEN 0x00000110UL
- #define CKM_RC4 0x00000111UL
- #define CKM_DES_KEY_GEN 0x00000120UL
- #define CKM_DES_ECB 0x00000121UL
- #define CKM_DES_CBC 0x00000122UL
- #define CKM_DES_MAC 0x00000123UL
- #define CKM_DES_MAC_GENERAL 0x00000124UL
- #define CKM_DES_CBC_PAD 0x00000125UL
- #define CKM_DES2_KEY_GEN 0x00000130UL
- #define CKM_DES3_KEY_GEN 0x00000131UL
- #define CKM_DES3_ECB 0x00000132UL
- #define CKM_DES3_CBC 0x00000133UL
- #define CKM_DES3_MAC 0x00000134UL
- #define CKM_DES3_MAC_GENERAL 0x00000135UL
- #define CKM_DES3_CBC_PAD 0x00000136UL
- #define CKM_DES3_CMAC_GENERAL 0x00000137UL
- #define CKM_DES3_CMAC 0x00000138UL
- #define CKM_CDMF_KEY_GEN 0x00000140UL
- #define CKM_CDMF_ECB 0x00000141UL
- #define CKM_CDMF_CBC 0x00000142UL
- #define CKM_CDMF_MAC 0x00000143UL
- #define CKM_CDMF_MAC_GENERAL 0x00000144UL

- #define CKM_CDMF_CBC_PAD 0x00000145UL
- #define CKM_DES_OFB64 0x00000150UL
- #define CKM_DES_OFB8 0x00000151UL
- #define CKM_DES_CFB64 0x00000152UL
- #define CKM_DES_CFB8 0x00000153UL
- #define CKM_MD2 0x00000200UL
- #define CKM_MD2_HMAC 0x00000201UL
- #define CKM_MD2_HMAC_GENERAL 0x00000202UL
- #define CKM_MD5 0x00000210UL
- #define CKM_MD5_HMAC 0x00000211UL
- #define CKM_MD5_HMAC_GENERAL 0x00000212UL
- #define CKM_SHA_1 0x00000220UL
- #define CKM_SHA_1_HMAC 0x00000221UL
- #define CKM_SHA_1_HMAC_GENERAL 0x00000222UL
- #define CKM_RIPEMD128 0x00000230UL
- #define CKM_RIPEMD128_HMAC 0x00000231UL
- #define CKM_RIPEMD128_HMAC_GENERAL 0x00000232UL
- #define CKM_RIPEMD160 0x00000240UL
- #define CKM_RIPEMD160_HMAC 0x00000241UL
- #define CKM_RIPEMD160_HMAC_GENERAL 0x00000242UL
- #define CKM_SHA256 0x00000250UL
- #define CKM_SHA256_HMAC 0x00000251UL
- #define CKM_SHA256_HMAC_GENERAL 0x00000252UL
- #define CKM_SHA224 0x00000255UL
- #define CKM_SHA224_HMAC 0x00000256UL
- #define CKM_SHA224_HMAC_GENERAL 0x00000257UL
- #define CKM_SHA384 0x00000260UL
- #define CKM_SHA384_HMAC 0x00000261UL
- #define CKM_SHA384_HMAC_GENERAL 0x00000262UL
- #define CKM_SHA512 0x00000270UL
- #define CKM_SHA512_HMAC 0x00000271UL
- #define CKM_SHA512_HMAC_GENERAL 0x00000272UL
- #define CKM_SECURID_KEY_GEN 0x00000280UL
- #define CKM_SECURID 0x00000282UL
- #define CKM_HOTP_KEY_GEN 0x00000290UL
- #define CKM_HOTP 0x00000291UL
- #define CKM_ACTI 0x000002A0UL
- #define CKM_ACTI_KEY_GEN 0x000002A1UL
- #define CKM_CAST_KEY_GEN 0x00000300UL
- #define CKM_CAST_ECB 0x00000301UL
- #define CKM_CAST_CBC 0x00000302UL
- #define CKM_CAST_MAC 0x00000303UL
- #define CKM_CAST_MAC_GENERAL 0x00000304UL
- #define CKM_CAST_CBC_PAD 0x00000305UL
- #define CKM_CAST3_KEY_GEN 0x00000310UL
- #define CKM_CAST3_ECB 0x00000311UL
- #define CKM_CAST3_CBC 0x00000312UL
- #define CKM_CAST3_MAC 0x00000313UL
- #define CKM_CAST3_MAC_GENERAL 0x00000314UL
- #define CKM_CAST3_CBC_PAD 0x00000315UL
- #define CKM_CAST5_KEY_GEN 0x00000320UL
- #define CKM_CAST128_KEY_GEN 0x00000320UL
- #define CKM_CAST5_ECB 0x00000321UL
- #define CKM_CAST128_ECB 0x00000321UL
- #define CKM_CAST5_CBC 0x00000322UL /* Deprecated */

- #define CKM_CAST128_CBC 0x00000322UL
- #define CKM_CAST5_MAC 0x00000323UL /* Deprecated */
- #define CKM_CAST128_MAC 0x00000323UL
- #define CKM_CAST5_MAC_GENERAL 0x00000324UL /* Deprecated */
- #define CKM_CAST128_MAC_GENERAL 0x00000324UL
- #define CKM_CAST5_CBC_PAD 0x00000325UL /* Deprecated */
- #define CKM_CAST128_CBC_PAD 0x00000325UL
- #define CKM_RC5_KEY_GEN 0x00000330UL
- #define CKM_RC5_ECB 0x00000331UL
- #define CKM_RC5_CBC 0x00000332UL
- #define CKM_RC5_MAC 0x00000333UL
- #define CKM_RC5_MAC_GENERAL 0x00000334UL
- #define CKM_RC5_CBC_PAD 0x00000335UL
- #define CKM_IDEA_KEY_GEN 0x00000340UL
- #define CKM_IDEA_ECB 0x00000341UL
- #define CKM_IDEA_CBC 0x00000342UL
- #define CKM_IDEA_MAC 0x00000343UL
- #define CKM_IDEA_MAC_GENERAL 0x00000344UL
- #define CKM_IDEA_CBC_PAD 0x00000345UL
- #define CKM_GENERIC_SECRET_KEY_GEN 0x00000350UL
- #define CKM_CONCATENATE_BASE_AND_KEY 0x00000360UL
- #define CKM_CONCATENATE_BASE_AND_DATA 0x00000362UL
- #define CKM_CONCATENATE_DATA_AND_BASE 0x00000363UL
- #define CKM_XOR_BASE_AND_DATA 0x00000364UL
- #define CKM_EXTRACT_KEY_FROM_KEY 0x00000365UL
- #define CKM_SSL3_PRE_MASTER_KEY_GEN 0x00000370UL
- #define CKM_SSL3_MASTER_KEY_DERIVE 0x00000371UL
- #define CKM_SSL3_KEY_AND_MAC_DERIVE 0x00000372UL
- #define CKM_SSL3_MASTER_KEY_DERIVE_DH 0x00000373UL
- #define CKM_TLS_PRE_MASTER_KEY_GEN 0x00000374UL
- #define CKM_TLS_MASTER_KEY_DERIVE 0x00000375UL
- #define CKM_TLS_KEY_AND_MAC_DERIVE 0x00000376UL
- #define CKM_TLS_MASTER_KEY_DERIVE_DH 0x00000377UL
- #define CKM_TLS_PRF 0x00000378UL
- #define CKM_SSL3_MD5_MAC 0x00000380UL
- #define CKM_SSL3_SHA1_MAC 0x00000381UL
- #define CKM_MD5_KEY_DERIVATION 0x00000390UL
- #define CKM_MD2_KEY_DERIVATION 0x00000391UL
- #define CKM_SHA1_KEY_DERIVATION 0x00000392UL
- #define CKM_SHA256_KEY_DERIVATION 0x00000393UL
- #define CKM_SHA384_KEY_DERIVATION 0x00000394UL
- #define CKM_SHA512_KEY_DERIVATION 0x00000395UL
- #define CKM_SHA224_KEY_DERIVATION 0x00000396UL
- #define CKM_PBE_MD2_DES_CBC 0x000003A0UL
- #define CKM_PBE_MD5_DES_CBC 0x000003A1UL
- #define CKM_PBE_MD5_CAST_CBC 0x000003A2UL
- #define CKM_PBE_MD5_CAST3_CBC 0x000003A3UL
- #define CKM_PBE_MD5_CAST5_CBC 0x000003A4UL /* Deprecated */
- #define CKM_PBE_MD5_CAST128_CBC 0x000003A4UL
- #define CKM_PBE_SHA1_CAST5_CBC 0x000003A5UL /* Deprecated */
- #define CKM_PBE_SHA1_CAST128_CBC 0x000003A5UL
- #define CKM_PBE_SHA1_RC4_128 0x000003A6UL
- #define CKM_PBE_SHA1_RC4_40 0x000003A7UL
- #define CKM_PBE_SHA1_DES3_EDE_CBC 0x000003A8UL
- #define CKM_PBE_SHA1_DES2_EDE_CBC 0x000003A9UL

- #define CKM_PBE_SHA1_RC2_128_CBC 0x000003AAUL
- #define CKM_PBE_SHA1_RC2_40_CBC 0x000003ABUL
- #define CKM_PKCS5_PBKD2 0x000003B0UL
- #define CKM_PBA_SHA1_WITH_SHA1_HMAC 0x000003C0UL
- #define CKM_WTLS_PRE_MASTER_KEY_GEN 0x000003D0UL
- #define CKM_WTLS_MASTER_KEY_DERIVE 0x000003D1UL
- #define CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC 0x000003D2UL
- #define CKM_WTLS_PRF 0x000003D3UL
- #define CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE 0x000003D4UL
- #define CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE 0x000003D5UL
- #define CKM_TLS10_MAC_SERVER 0x000003D6UL
- #define CKM_TLS10_MAC_CLIENT 0x000003D7UL
- #define CKM_TLS12_MAC 0x000003D8UL
- #define CKM_TLS12_KDF 0x000003D9UL
- #define CKM_TLS12_MASTER_KEY_DERIVE 0x000003E0UL
- #define CKM_TLS12_KEY_AND_MAC_DERIVE 0x000003E1UL
- #define CKM_TLS12_MASTER_KEY_DERIVE_DH 0x000003E2UL
- #define CKM_TLS12_KEY_SAFE_DERIVE 0x000003E3UL
- #define CKM_TLS_MAC 0x000003E4UL
- #define CKM_TLS_KDF 0x000003E5UL
- #define CKM_KEY_WRAP_LYNKS 0x00000400UL
- #define CKM_KEY_WRAP_SET_OAEP 0x00000401UL
- #define CKM_CMS_SIG 0x00000500UL
- #define CKM_KIP_DERIVE 0x00000510UL
- #define CKM_KIP_WRAP 0x00000511UL
- #define CKM_KIP_MAC 0x00000512UL
- #define CKM_CAMELLIA_KEY_GEN 0x00000550UL
- #define CKM_CAMELLIA_ECB 0x00000551UL
- #define CKM_CAMELLIA_CBC 0x00000552UL
- #define CKM_CAMELLIA_MAC 0x00000553UL
- #define CKM_CAMELLIA_MAC_GENERAL 0x00000554UL
- #define CKM_CAMELLIA_CBC_PAD 0x00000555UL
- #define CKM_CAMELLIA_ECB_ENCRYPT_DATA 0x00000556UL
- #define CKM_CAMELLIA_CBC_ENCRYPT_DATA 0x00000557UL
- #define CKM_CAMELLIA_CTR 0x00000558UL
- #define CKM_ARIA_KEY_GEN 0x00000560UL
- #define CKM_ARIA_ECB 0x00000561UL
- #define CKM_ARIA_CBC 0x00000562UL
- #define CKM_ARIA_MAC 0x00000563UL
- #define CKM_ARIA_MAC_GENERAL 0x00000564UL
- #define CKM_ARIA_CBC_PAD 0x00000565UL
- #define CKM_ARIA_ECB_ENCRYPT_DATA 0x00000566UL
- #define CKM_ARIA_CBC_ENCRYPT_DATA 0x00000567UL
- #define CKM_SEED_KEY_GEN 0x00000650UL
- #define CKM_SEED_ECB 0x00000651UL
- #define CKM_SEED_CBC 0x00000652UL
- #define CKM_SEED_MAC 0x00000653UL
- #define CKM_SEED_MAC_GENERAL 0x00000654UL
- #define CKM_SEED_CBC_PAD 0x00000655UL
- #define CKM_SEED_ECB_ENCRYPT_DATA 0x00000656UL
- #define CKM_SEED_CBC_ENCRYPT_DATA 0x00000657UL
- #define CKM_SKIPJACK_KEY_GEN 0x00001000UL
- #define CKM_SKIPJACK_ECB64 0x00001001UL
- #define CKM_SKIPJACK_CBC64 0x00001002UL
- #define CKM_SKIPJACK_OFB64 0x00001003UL

- #define CKM_SKIPJACK_CFB64 0x00001004UL
- #define CKM_SKIPJACK_CFB32 0x00001005UL
- #define CKM_SKIPJACK_CFB16 0x00001006UL
- #define CKM_SKIPJACK_CFB8 0x00001007UL
- #define CKM_SKIPJACK_WRAP 0x00001008UL
- #define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009UL
- #define CKM_SKIPJACK_RELAYX 0x0000100aUL
- #define CKM_KEA_KEY_PAIR_GEN 0x00001010UL
- #define CKM_KEA_KEY_DERIVE 0x00001011UL
- #define CKM_KEA_DERIVE 0x00001012UL
- #define CKM_FORTEZZA_TIMESTAMP 0x00001020UL
- #define CKM_BATON_KEY_GEN 0x00001030UL
- #define CKM_BATON_ECB128 0x00001031UL
- #define CKM_BATON_ECB96 0x00001032UL
- #define CKM_BATON_CBC128 0x00001033UL
- #define CKM_BATON_COUNTER 0x00001034UL
- #define CKM_BATON_SHUFFLE 0x00001035UL
- #define CKM_BATON_WRAP 0x00001036UL
- #define CKM_ECDSA_KEY_PAIR_GEN 0x00001040UL /* Deprecated */
- #define CKM_EC_KEY_PAIR_GEN 0x00001040UL
- #define CKM_ECDSA 0x00001041UL
- #define CKM_ECDSA_SHA1 0x00001042UL
- #define CKM_ECDSA_SHA224 0x00001043UL
- #define CKM_ECDSA_SHA256 0x00001044UL
- #define CKM_ECDSA_SHA384 0x00001045UL
- #define CKM_ECDSA_SHA512 0x00001046UL
- #define CKM_ECDH1_DERIVE 0x00001050UL
- #define CKM_ECDH1_COFACTOR_DERIVE 0x00001051UL
- #define CKM_ECMQV_DERIVE 0x00001052UL
- #define CKM_ECDH_AES_KEY_WRAP 0x00001053UL
- #define CKM_RSA_AES_KEY_WRAP 0x00001054UL
- #define CKM_JUNIPER_KEY_GEN 0x00001060UL
- #define CKM_JUNIPER_ECB128 0x00001061UL
- #define CKM_JUNIPER_CBC128 0x00001062UL
- #define CKM_JUNIPER_COUNTER 0x00001063UL
- #define CKM_JUNIPER_SHUFFLE 0x00001064UL
- #define CKM_JUNIPER_WRAP 0x00001065UL
- #define CKM_FASTHASH 0x00001070UL
- #define CKM_AES_KEY_GEN 0x00001080UL
- #define CKM_AES_ECB 0x00001081UL
- #define CKM_AES_CBC 0x00001082UL
- #define CKM_AES_MAC 0x00001083UL
- #define CKM_AES_MAC_GENERAL 0x00001084UL
- #define CKM_AES_CBC_PAD 0x00001085UL
- #define CKM_AES_CTR 0x00001086UL
- #define CKM_AES_GCM 0x00001087UL
- #define CKM_AES_CCM 0x00001088UL
- #define CKM_AES_CTS 0x00001089UL
- #define CKM_AES_CMAC 0x0000108AUL
- #define CKM_AES_CMAC_GENERAL 0x0000108BUL
- #define CKM_AES_XCBC_MAC 0x0000108CUL
- #define CKM_AES_XCBC_MAC_96 0x0000108DUL
- #define CKM_AES_GMAC 0x0000108EUL
- #define CKM_BLOWFISH_KEY_GEN 0x00001090UL
- #define CKM_BLOWFISH_CBC 0x00001091UL

- #define CKM_TWOFISH_KEY_GEN 0x00001092UL
- #define CKM_TWOFISH_CBC 0x00001093UL
- #define CKM_BLOWFISH_CBC_PAD 0x00001094UL
- #define CKM_TWOFISH_CBC_PAD 0x00001095UL
- #define CKM_DES_ECB_ENCRYPT_DATA 0x00001100UL
- #define CKM_DES_CBC_ENCRYPT_DATA 0x00001101UL
- #define CKM_DES3_ECB_ENCRYPT_DATA 0x00001102UL
- #define CKM_DES3_CBC_ENCRYPT_DATA 0x00001103UL
- #define CKM_AES_ECB_ENCRYPT_DATA 0x00001104UL
- #define CKM_AES_CBC_ENCRYPT_DATA 0x00001105UL
- #define CKM_GOSTR3410_KEY_PAIR_GEN 0x00001200UL
- #define CKM_GOSTR3410 0x00001201UL
- #define CKM_GOSTR3410_WITH_GOSTR3411 0x00001202UL
- #define CKM_GOSTR3410_KEY_WRAP 0x00001203UL
- #define CKM_GOSTR3410_DERIVE 0x00001204UL
- #define CKM_GOSTR3411 0x00001210UL
- #define CKM_GOSTR3411_HMAC 0x00001211UL
- #define CKM_GOST28147_KEY_GEN 0x00001220UL
- #define CKM_GOST28147_ECB 0x00001221UL
- #define CKM_GOST28147 0x00001222UL
- #define CKM_GOST28147_MAC 0x00001223UL
- #define CKM_GOST28147_KEY_WRAP 0x00001224UL
- #define CKM_DSA_PARAMETER_GEN 0x00002000UL
- #define CKM_DH_PKCS_PARAMETER_GEN 0x00002001UL
- #define CKM_X9_42_DH_PARAMETER_GEN 0x00002002UL
- #define CKM_DSA_PROBABLISTIC_PARAMETER_GEN 0x00002003UL
- #define CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN 0x00002004UL
- #define CKM_AES_OFB 0x00002104UL
- #define CKM_AES_CFB64 0x00002105UL
- #define CKM_AES_CFB8 0x00002106UL
- #define CKM_AES_CFB128 0x00002107UL
- #define CKM_AES_CFB1 0x00002108UL
- #define CKM_AES_KEY_WRAP 0x00002109UL /∗ WAS: 0x00001090 ∗/
- #define CKM_AES_KEY_WRAP_PAD 0x0000210AUL /∗ WAS: 0x00001091 ∗/
- #define CKM_RSA_PKCS_TPM_1_1 0x00004001UL
- #define CKM_RSA_PKCS_OAEP_TPM_1_1 0x00004002UL
- #define CKM_VENDOR_DEFINED 0x80000000UL
- #define CKF_HW 0x00000001UL /∗ performed by HW ∗/
- #define CKF_ENCRYPT 0x00000100UL
- #define CKF_DECRYPT 0x00000200UL
- #define CKF_DIGEST 0x00000400UL
- #define CKF_SIGN 0x00000800UL
- #define CKF_SIGN_RECOVER 0x00001000UL
- #define CKF_VERIFY 0x00002000UL
- #define CKF_VERIFY_RECOVER 0x00004000UL
- #define CKF_GENERATE 0x00008000UL
- #define CKF_GENERATE_KEY_PAIR 0x00010000UL
- #define CKF_WRAP 0x00020000UL
- #define CKF_UNWRAP 0x00040000UL
- #define CKF_DERIVE 0x00080000UL
- #define CKF_EC_F_P 0x00100000UL
- #define CKF_EC_F_2M 0x00200000UL
- #define CKF_EC_ECPARAMETERS 0x00400000UL
- #define CKF_EC_NAMEDCURVE 0x00800000UL
- #define CKF_EC_UNCOMPRESS 0x01000000UL

- #define CKF_EC_COMPRESS 0x02000000UL
- #define CKF_EXTENSION 0x80000000UL
- #define CKR_OK 0x00000000UL
- #define CKR_CANCEL 0x00000001UL
- #define CKR_HOST_MEMORY 0x00000002UL
- #define CKR_SLOT_ID_INVALID 0x00000003UL
- #define CKR_GENERAL_ERROR 0x00000005UL
- #define CKR_FUNCTION_FAILED 0x00000006UL
- #define CKR_ARGUMENTS_BAD 0x00000007UL
- #define CKR_NO_EVENT 0x00000008UL
- #define CKR_NEED_TO_CREATE_THREADS 0x00000009UL
- #define CKR_CANT_LOCK 0x0000000AUL
- #define CKR_ATTRIBUTE_READ_ONLY 0x00000010UL
- #define CKR_ATTRIBUTE_SENSITIVE 0x00000011UL
- #define CKR_ATTRIBUTE_TYPE_INVALID 0x00000012UL
- #define CKR_ATTRIBUTE_VALUE_INVALID 0x00000013UL
- #define CKR_ACTION_PROHIBITED 0x0000001BUL
- #define CKR_DATA_INVALID 0x00000020UL
- #define CKR_DATA_LEN_RANGE 0x00000021UL
- #define CKR_DEVICE_ERROR 0x00000030UL
- #define CKR_DEVICE_MEMORY 0x00000031UL
- #define CKR_DEVICE_REMOVED 0x00000032UL
- #define CKR_ENCRYPTED_DATA_INVALID 0x00000040UL
- #define CKR_ENCRYPTED_DATA_LEN_RANGE 0x00000041UL
- #define CKR_FUNCTION_CANCELED 0x00000050UL
- #define CKR_FUNCTION_NOT_PARALLEL 0x00000051UL
- #define CKR_FUNCTION_NOT_SUPPORTED 0x00000054UL
- #define CKR_KEY_HANDLE_INVALID 0x00000060UL
- #define CKR_KEY_SIZE_RANGE 0x00000062UL
- #define CKR_KEY_TYPE_INCONSISTENT 0x00000063UL
- #define CKR_KEY_NOT_NEEDED 0x00000064UL
- #define CKR_KEY_CHANGED 0x00000065UL
- #define CKR_KEY_NEEDED 0x00000066UL
- #define CKR_KEY_INDIGESTIBLE 0x00000067UL
- #define CKR_KEY_FUNCTION_NOT_PERMITTED 0x00000068UL
- #define CKR_KEY_NOT_WRAPPABLE 0x00000069UL
- #define CKR_KEY_UNEXTRACTABLE 0x0000006AUL
- #define CKR_MECHANISM_INVALID 0x00000070UL
- #define CKR_MECHANISM_PARAM_INVALID 0x00000071UL
- #define CKR_OBJECT_HANDLE_INVALID 0x00000082UL
- #define CKR_OPERATION_ACTIVE 0x00000090UL
- #define CKR_OPERATION_NOT_INITIALIZED 0x00000091UL
- #define CKR_PIN_INCORRECT 0x000000A0UL
- #define CKR_PIN_INVALID 0x000000A1UL
- #define CKR_PIN_LEN_RANGE 0x000000A2UL
- #define CKR_PIN_EXPIRED 0x000000A3UL
- #define CKR_PIN_LOCKED 0x000000A4UL
- #define CKR_SESSION_CLOSED 0x000000B0UL
- #define CKR_SESSION_COUNT 0x000000B1UL
- #define CKR_SESSION_HANDLE_INVALID 0x000000B3UL
- #define CKR_SESSION_PARALLEL_NOT_SUPPORTED 0x000000B4UL
- #define CKR_SESSION_READ_ONLY 0x000000B5UL
- #define CKR_SESSION_EXISTS 0x000000B6UL
- #define CKR_SESSION_READ_ONLY_EXISTS 0x000000B7UL
- #define CKR_SESSION_READ_WRITE_SO_EXISTS 0x000000B8UL

- #define CKR_SIGNATURE_INVALID 0x000000C0UL
- #define CKR_SIGNATURE_LEN_RANGE 0x000000C1UL
- #define CKR_TEMPLATE_INCOMPLETE 0x000000D0UL
- #define CKR_TEMPLATE_INCONSISTENT 0x000000D1UL
- #define CKR_TOKEN_NOT_PRESENT 0x000000E0UL
- #define CKR_TOKEN_NOT_RECOGNIZED 0x000000E1UL
- #define CKR_TOKEN_WRITE_PROTECTED 0x000000E2UL
- #define CKR_UNWRAPPING_KEY_HANDLE_INVALID 0x000000F0UL
- #define CKR_UNWRAPPING_KEY_SIZE_RANGE 0x000000F1UL
- #define CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT 0x000000F2UL
- #define CKR_USER_ALREADY_LOGGED_IN 0x00000100UL
- #define CKR_USER_NOT_LOGGED_IN 0x00000101UL
- #define CKR_USER_PIN_NOT_INITIALIZED 0x00000102UL
- #define CKR_USER_TYPE_INVALID 0x00000103UL
- #define CKR_USER_ANOTHER_ALREADY_LOGGED_IN 0x00000104UL
- #define CKR_USER_TOO_MANY_TYPES 0x00000105UL
- #define CKR_WRAPPED_KEY_INVALID 0x00000110UL
- #define CKR_WRAPPED_KEY_LEN_RANGE 0x00000112UL
- #define CKR_WRAPPING_KEY_HANDLE_INVALID 0x00000113UL
- #define CKR_WRAPPING_KEY_SIZE_RANGE 0x00000114UL
- #define CKR_WRAPPING_KEY_TYPE_INCONSISTENT 0x00000115UL
- #define CKR_RANDOM_SEED_NOT_SUPPORTED 0x00000120UL
- #define CKR_RANDOM_NO_RNG 0x00000121UL
- #define CKR_DOMAIN_PARAMS_INVALID 0x00000130UL
- #define CKR_CURVE_NOT_SUPPORTED 0x00000140UL
- #define CKR_BUFFER_TOO_SMALL 0x00000150UL
- #define CKR_SAVED_STATE_INVALID 0x00000160UL
- #define CKR_INFORMATION_SENSITIVE 0x00000170UL
- #define CKR_STATE_UNSAVEABLE 0x00000180UL
- #define CKR_CRYPTOKI_NOT_INITIALIZED 0x00000190UL
- #define CKR_CRYPTOKI_ALREADY_INITIALIZED 0x00000191UL
- #define CKR_MUTEX_BAD 0x000001A0UL
- #define CKR_MUTEX_NOT_LOCKED 0x000001A1UL
- #define CKR_NEW_PIN_MODE 0x000001B0UL
- #define CKR_NEXT_OTP 0x000001B1UL
- #define CKR_EXCEEDED_MAX_ITERATIONS 0x000001B5UL
- #define CKR_FIPS_SELF_TEST_FAILED 0x000001B6UL
- #define CKR_LIBRARY_LOAD_FAILED 0x000001B7UL
- #define CKR_PIN_TOO_WEAK 0x000001B8UL
- #define CKR_PUBLIC_KEY_INVALID 0x000001B9UL
- #define CKR_FUNCTION_REJECTED 0x00000200UL
- #define CKR_VENDOR_DEFINED 0x80000000UL
- #define CKF_LIBRARY_CANT_CREATE_OS_THREADS 0x00000001UL
- #define CKF_OS_LOCKING_OK 0x00000002UL
- #define CKF_DONT_BLOCK 1
- #define CKG_MGF1_SHA1 0x00000001UL
- #define CKG_MGF1_SHA256 0x00000002UL
- #define CKG_MGF1_SHA384 0x00000003UL
- #define CKG_MGF1_SHA512 0x00000004UL
- #define CKG_MGF1_SHA224 0x00000005UL
- #define CKZ_DATA_SPECIFIED 0x00000001UL
- #define CKD_NULL 0x00000001UL
- #define CKD_SHA1_KDF 0x00000002UL
- #define CKD_SHA1_KDF_ASN1 0x00000003UL
- #define CKD_SHA1_KDF_CONCATENATE 0x00000004UL

- #define CKD_SHA224_KDF 0x00000005UL
- #define CKD_SHA256_KDF 0x00000006UL
- #define CKD_SHA384_KDF 0x00000007UL
- #define CKD_SHA512_KDF 0x00000008UL
- #define CKD_CPDIVERSIFY_KDF 0x00000009UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA1 0x00000001UL
- #define CKP_PKCS5_PBKD2_HMAC_GOSTR3411 0x00000002UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA224 0x00000003UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA256 0x00000004UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA384 0x00000005UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA512 0x00000006UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA512_224 0x00000007UL
- #define CKP_PKCS5_PBKD2_HMAC_SHA512_256 0x00000008UL
- #define CKZ_SALT_SPECIFIED 0x00000001UL
- #define CK_OTP_VALUE 0UL
- #define CK_OTP_PIN 1UL
- #define CK_OTP_CHALLENGE 2UL
- #define CK_OTP_TIME 3UL
- #define CK_OTP_COUNTER 4UL
- #define CK_OTP_FLAGS 5UL
- #define CK_OTP_OUTPUT_LENGTH 6UL
- #define CK_OTP_OUTPUT_FORMAT 7UL
- #define CKF_NEXT_OTP 0x00000001UL
- #define CKF_EXCLUDE_TIME 0x00000002UL
- #define CKF_EXCLUDE_COUNTER 0x00000004UL
- #define CKF_EXCLUDE_CHALLENGE 0x00000008UL
- #define CKF_EXCLUDE_PIN 0x00000010UL
- #define CKF_USER_FRIENDLY_OTP 0x00000020UL

Typedefs

- typedef unsigned char CK_BYTE
- typedef CK_BYTE CK_CHAR
- typedef CK_BYTE CK_UTF8CHAR
- typedef CK_BYTE CK_BBOOL
- typedef unsigned long int CK_ULONG
- typedef long int CK_LONG
- typedef CK_ULONG CK_FLAGS
- typedef CK_BYTE CK_PTR CK_BYTE_PTR
- typedef CK_CHAR CK_PTR CK_CHAR_PTR
- typedef CK_UTF8CHAR CK_PTR CK_UTF8CHAR_PTR
- typedef CK_ULONG CK_PTR CK_ULONG_PTR
- typedef void CK_PTR CK_VOID_PTR
- typedef CK_VOID_PTR CK_PTR CK_VOID_PTR_PTR
- typedef struct CK_VERSION CK_VERSION
- typedef CK_VERSION CK_PTR CK_VERSION_PTR
- typedef struct CK_INFO CK_INFO
- typedef CK_INFO CK_PTR CK_INFO_PTR
- typedef CK_ULONG CK_NOTIFICATION
- typedef CK_ULONG CK_SLOT_ID
- typedef CK_SLOT_ID CK_PTR CK_SLOT_ID_PTR
- typedef struct CK_SLOT_INFO CK_SLOT_INFO
- typedef CK_SLOT_INFO CK_PTR CK_SLOT_INFO_PTR
- typedef struct CK_TOKEN_INFO CK_TOKEN_INFO

- typedef CK_TOKEN_INFO CK_PTR CK_TOKEN_INFO_PTR
- typedef CK_ULONG CK_SESSION_HANDLE
- typedef CK_SESSION_HANDLE CK_PTR CK_SESSION_HANDLE_PTR
- typedef CK_ULONG CK_USER_TYPE
- typedef CK_ULONG CK_STATE
- typedef struct CK_SESSION_INFO CK_SESSION_INFO
- typedef CK_SESSION_INFO CK_PTR CK_SESSION_INFO_PTR
- typedef CK_ULONG CK_OBJECT_HANDLE
- typedef CK_OBJECT_HANDLE CK_PTR CK_OBJECT_HANDLE_PTR
- typedef CK_ULONG CK_OBJECT_CLASS
- typedef CK_OBJECT_CLASS CK_PTR CK_OBJECT_CLASS_PTR
- typedef CK_ULONG CK_HW_FEATURE_TYPE
- typedef CK_ULONG CK_KEY_TYPE
- typedef CK_ULONG CK_CERTIFICATE_TYPE
- typedef CK_ULONG CK_ATTRIBUTE_TYPE
- typedef struct CK_ATTRIBUTE CK_ATTRIBUTE
- typedef CK_ATTRIBUTE CK_PTR CK_ATTRIBUTE_PTR
- typedef struct CK_DATE CK_DATE
- typedef CK_ULONG CK_MECHANISM_TYPE
- typedef CK_MECHANISM_TYPE CK_PTR CK_MECHANISM_TYPE_PTR
- typedef struct CK_MECHANISM CK_MECHANISM
- typedef CK_MECHANISM CK_PTR CK_MECHANISM_PTR
- typedef struct CK_MECHANISM_INFO CK_MECHANISM_INFO
- typedef CK_MECHANISM_INFO CK_PTR CK_MECHANISM_INFO_PTR
- typedef CK_ULONG CK_RV
- typedef CK_NOTIFICATION event
- typedef CK_NOTIFICATION CK_VOID_PTR pApplication
- typedef struct CK_FUNCTION_LIST CK_FUNCTION_LIST
- typedef CK_FUNCTION_LIST CK_PTR CK_FUNCTION_LIST_PTR
- typedef CK_FUNCTION_LIST_PTR CK_PTR CK_FUNCTION_LIST_PTR_PTR
- typedef struct CK_C_INITIALIZE_ARGS CK_C_INITIALIZE_ARGS
- typedef CK_C_INITIALIZE_ARGS CK_PTR CK_C_INITIALIZE_ARGS_PTR
- typedef CK_ULONG CK_RSA_PKCS_MGF_TYPE
- typedef CK_RSA_PKCS_MGF_TYPE CK_PTR CK_RSA_PKCS_MGF_TYPE_PTR
- typedef CK_ULONG CK_RSA_PKCS_OAEP_SOURCE_TYPE
- typedef CK_RSA_PKCS_OAEP_SOURCE_TYPE CK_PTR CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR
- typedef struct CK_RSA_PKCS_OAEP_PARAMS CK_RSA_PKCS_OAEP_PARAMS
- typedef CK_RSA_PKCS_OAEP_PARAMS CK_PTR CK_RSA_PKCS_OAEP_PARAMS_PTR
- typedef struct CK_RSA_PKCS_PSS_PARAMS CK_RSA_PKCS_PSS_PARAMS
- typedef CK_RSA_PKCS_PSS_PARAMS CK_PTR CK_RSA_PKCS_PSS_PARAMS_PTR
- typedef CK_ULONG CK_EC_KDF_TYPE
- typedef struct CK_ECDH1_DERIVE_PARAMS CK_ECDH1_DERIVE_PARAMS
- typedef CK_ECDH1_DERIVE_PARAMS CK_PTR CK_ECDH1_DERIVE_PARAMS_PTR
- typedef struct CK_ECDH2_DERIVE_PARAMS CK_ECDH2_DERIVE_PARAMS
- typedef CK_ECDH2_DERIVE_PARAMS CK_PTR CK_ECDH2_DERIVE_PARAMS_PTR
- typedef struct CK_ECMQV_DERIVE_PARAMS CK_ECMQV_DERIVE_PARAMS
- typedef CK_ECMQV_DERIVE_PARAMS CK_PTR CK_ECMQV_DERIVE_PARAMS_PTR
- typedef CK_ULONG CK_X9_42_DH_KDF_TYPE
- typedef CK_X9_42_DH_KDF_TYPE CK_PTR CK_X9_42_DH_KDF_TYPE_PTR
- typedef struct CK_X9_42_DH1_DERIVE_PARAMS CK_X9_42_DH1_DERIVE_PARAMS
- typedef struct CK_X9_42_DH1_DERIVE_PARAMS CK_PTR CK_X9_42_DH1_DERIVE_PARAMS_PTR
- typedef struct CK_X9_42_DH2_DERIVE_PARAMS CK_X9_42_DH2_DERIVE_PARAMS
- typedef CK_X9_42_DH2_DERIVE_PARAMS CK_PTR CK_X9_42_DH2_DERIVE_PARAMS_PTR
- typedef struct CK_X9_42_MQV_DERIVE_PARAMS CK_X9_42_MQV_DERIVE_PARAMS
- typedef CK_X9_42_MQV_DERIVE_PARAMS CK_PTR CK_X9_42_MQV_DERIVE_PARAMS_PTR

- typedef struct CK_KEA_DERIVE_PARAMS CK_KEA_DERIVE_PARAMS
- typedef CK_KEA_DERIVE_PARAMS CK_PTR CK_KEA_DERIVE_PARAMS_PTR
- typedef CK_ULONG CK_RC2_PARAMS
- typedef CK_RC2_PARAMS CK_PTR CK_RC2_PARAMS_PTR
- typedef struct CK_RC2_CBC_PARAMS CK_RC2_CBC_PARAMS
- typedef CK_RC2_CBC_PARAMS CK_PTR CK_RC2_CBC_PARAMS_PTR
- typedef struct CK_RC2_MAC_GENERAL_PARAMS CK_RC2_MAC_GENERAL_PARAMS
- typedef CK_RC2_MAC_GENERAL_PARAMS CK_PTR CK_RC2_MAC_GENERAL_PARAMS_PTR
- typedef struct CK_RC5_PARAMS CK_RC5_PARAMS
- typedef CK_RC5_PARAMS CK_PTR CK_RC5_PARAMS_PTR
- typedef struct CK_RC5_CBC_PARAMS CK_RC5_CBC_PARAMS
- typedef CK_RC5_CBC_PARAMS CK_PTR CK_RC5_CBC_PARAMS_PTR
- typedef struct CK_RC5_MAC_GENERAL_PARAMS CK_RC5_MAC_GENERAL_PARAMS
- typedef CK_RC5_MAC_GENERAL_PARAMS CK_PTR CK_RC5_MAC_GENERAL_PARAMS_PTR
- typedef CK_ULONG CK_MAC_GENERAL_PARAMS
- typedef CK_MAC_GENERAL_PARAMS CK_PTR CK_MAC_GENERAL_PARAMS_PTR
- typedef struct CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_DES_CBC_ENCRYPT_DATA_PARAMS
- typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR
- typedef struct CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_AES_CBC_ENCRYPT_DATA_PARAMS
- typedef CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_AES_CBC_ENCRYPT_DATA_PARAMS_PTR
- typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS CK_SKIPJACK_PRIVATE_WRAP_PARAMS
- typedef CK_SKIPJACK_PRIVATE_WRAP_PARAMS CK_PTR CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR
- typedef struct CK_SKIPJACK_RELAYX_PARAMS CK_SKIPJACK_RELAYX_PARAMS
- typedef CK_SKIPJACK_RELAYX_PARAMS CK_PTR CK_SKIPJACK_RELAYX_PARAMS_PTR
- typedef struct CK_PBE_PARAMS CK_PBE_PARAMS
- typedef CK_PBE_PARAMS CK_PTR CK_PBE_PARAMS_PTR
- typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS CK_KEY_WRAP_SET_OAEP_PARAMS
- typedef CK_KEY_WRAP_SET_OAEP_PARAMS CK_PTR CK_KEY_WRAP_SET_OAEP_PARAMS_PTR
- typedef struct CK_SSL3_RANDOM_DATA CK_SSL3_RANDOM_DATA
- typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS CK_SSL3_MASTER_KEY_DERIVE_PARAMS
- typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR
- typedef struct CK_SSL3_KEY_MAT_OUT CK_SSL3_KEY_MAT_OUT
- typedef CK_SSL3_KEY_MAT_OUT CK_PTR CK_SSL3_KEY_MAT_OUT_PTR
- typedef struct CK_SSL3_KEY_MAT_PARAMS CK_SSL3_KEY_MAT_PARAMS
- typedef CK_SSL3_KEY_MAT_PARAMS CK_PTR CK_SSL3_KEY_MAT_PARAMS_PTR
- typedef struct CK_TLS_PRF_PARAMS CK_TLS_PRF_PARAMS
- typedef CK_TLS_PRF_PARAMS CK_PTR CK_TLS_PRF_PARAMS_PTR
- typedef struct CK_WTLS_RANDOM_DATA CK_WTLS_RANDOM_DATA
- typedef CK_WTLS_RANDOM_DATA CK_PTR CK_WTLS_RANDOM_DATA_PTR
- typedef struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS CK_WTLS_MASTER_KEY_DERIVE_PARAMS
- typedef CK_WTLS_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR
- typedef struct CK_WTLS_PRF_PARAMS CK_WTLS_PRF_PARAMS
- typedef CK_WTLS_PRF_PARAMS CK_PTR CK_WTLS_PRF_PARAMS_PTR
- typedef struct CK_WTLS_KEY_MAT_OUT CK_WTLS_KEY_MAT_OUT
- typedef CK_WTLS_KEY_MAT_OUT CK_PTR CK_WTLS_KEY_MAT_OUT_PTR
- typedef struct CK_WTLS_KEY_MAT_PARAMS CK_WTLS_KEY_MAT_PARAMS
- typedef CK_WTLS_KEY_MAT_PARAMS CK_PTR CK_WTLS_KEY_MAT_PARAMS_PTR
- typedef struct CK_CMS_SIG_PARAMS CK_CMS_SIG_PARAMS
- typedef CK_CMS_SIG_PARAMS CK_PTR CK_CMS_SIG_PARAMS_PTR
- typedef struct CK_KEY_DERIVATION_STRING_DATA CK_KEY_DERIVATION_STRING_DATA
- typedef CK_KEY_DERIVATION_STRING_DATA CK_PTR CK_KEY_DERIVATION_STRING_DATA_PTR
- typedef CK_ULONG CK_EXTRACT_PARAMS
- typedef CK_EXTRACT_PARAMS CK_PTR CK_EXTRACT_PARAMS_PTR
- typedef CK_ULONG CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE
- typedef CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE CK_PTR CK_PKCS5_PBKD2_PSEUDO_RANDOM_F

- typedef CK_ULONG CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE
- typedef CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE CK_PTR CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE_PTR
- typedef struct CK_PKCS5_PBKD2_PARAMS CK_PKCS5_PBKD2_PARAMS
- typedef CK_PKCS5_PBKD2_PARAMS CK_PTR CK_PKCS5_PBKD2_PARAMS_PTR
- typedef struct CK_PKCS5_PBKD2_PARAMS2 CK_PKCS5_PBKD2_PARAMS2
- typedef CK_PKCS5_PBKD2_PARAMS2 CK_PTR CK_PKCS5_PBKD2_PARAMS2_PTR
- typedef CK_ULONG CK_OTP_PARAM_TYPE
- typedef CK_OTP_PARAM_TYPE CK_PARAM_TYPE
- typedef struct CK_OTP_PARAM CK_OTP_PARAM
- typedef CK_OTP_PARAM CK_PTR CK_OTP_PARAM_PTR
- typedef struct CK_OTP_PARAMS CK_OTP_PARAMS
- typedef CK_OTP_PARAMS CK_PTR CK_OTP_PARAMS_PTR
- typedef struct CK_OTP_SIGNATURE_INFO CK_OTP_SIGNATURE_INFO
- typedef CK_OTP_SIGNATURE_INFO CK_PTR CK_OTP_SIGNATURE_INFO_PTR
- typedef struct CK_KIP_PARAMS CK_KIP_PARAMS
- typedef CK_KIP_PARAMS CK_PTR CK_KIP_PARAMS_PTR
- typedef struct CK_AES_CTR_PARAMS CK_AES_CTR_PARAMS
- typedef CK_AES_CTR_PARAMS CK_PTR CK_AES_CTR_PARAMS_PTR
- typedef struct CK_GCM_PARAMS CK_GCM_PARAMS
- typedef CK_GCM_PARAMS CK_PTR CK_GCM_PARAMS_PTR
- typedef struct CK_CCM_PARAMS CK_CCM_PARAMS
- typedef CK_CCM_PARAMS CK_PTR CK_CCM_PARAMS_PTR
- typedef struct CK_AES_GCM_PARAMS CK_AES_GCM_PARAMS
- typedef CK_AES_GCM_PARAMS CK_PTR CK_AES_GCM_PARAMS_PTR
- typedef struct CK_AES_CCM_PARAMS CK_AES_CCM_PARAMS
- typedef CK_AES_CCM_PARAMS CK_PTR CK_AES_CCM_PARAMS_PTR
- typedef struct CK_CAMELLIA_CTR_PARAMS CK_CAMELLIA_CTR_PARAMS
- typedef CK_CAMELLIA_CTR_PARAMS CK_PTR CK_CAMELLIA_CTR_PARAMS_PTR
- typedef struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS
- typedef CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS_PT
- typedef struct CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_ARIA_CBC_ENCRYPT_DATA_PARAMS
- typedef CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_ARIA_CBC_ENCRYPT_DATA_PARAMS_PTR
- typedef struct CK_DSA_PARAMETER_GEN_PARAM CK_DSA_PARAMETER_GEN_PARAM
- typedef CK_DSA_PARAMETER_GEN_PARAM CK_PTR CK_DSA_PARAMETER_GEN_PARAM_PTR
- typedef struct CK_ECDH_AES_KEY_WRAP_PARAMS CK_ECDH_AES_KEY_WRAP_PARAMS
- typedef CK_ECDH_AES_KEY_WRAP_PARAMS CK_PTR CK_ECDH_AES_KEY_WRAP_PARAMS_PTR
- typedef CK_ULONG CK_JAVA_MIDP_SECURITY_DOMAIN
- typedef CK_ULONG CK_CERTIFICATE_CATEGORY
- typedef struct CK_RSA_AES_KEY_WRAP_PARAMS CK_RSA_AES_KEY_WRAP_PARAMS
- typedef CK_RSA_AES_KEY_WRAP_PARAMS CK_PTR CK_RSA_AES_KEY_WRAP_PARAMS_PTR
- typedef struct CK_TLS12_MASTER_KEY_DERIVE_PARAMS CK_TLS12_MASTER_KEY_DERIVE_PARAMS
- typedef CK_TLS12_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_TLS12_MASTER_KEY_DERIVE_PARAMS_PTR
- typedef struct CK_TLS12_KEY_MAT_PARAMS CK_TLS12_KEY_MAT_PARAMS
- typedef CK_TLS12_KEY_MAT_PARAMS CK_PTR CK_TLS12_KEY_MAT_PARAMS_PTR
- typedef struct CK_TLS_KDF_PARAMS CK_TLS_KDF_PARAMS
- typedef CK_TLS_KDF_PARAMS CK_PTR CK_TLS_KDF_PARAMS_PTR
- typedef struct CK_TLS_MAC_PARAMS CK_TLS_MAC_PARAMS
- typedef CK_TLS_MAC_PARAMS CK_PTR CK_TLS_MAC_PARAMS_PTR
- typedef struct CK_GOSTR3410_DERIVE_PARAMS CK_GOSTR3410_DERIVE_PARAMS
- typedef CK_GOSTR3410_DERIVE_PARAMS CK_PTR CK_GOSTR3410_DERIVE_PARAMS_PTR
- typedef struct CK_GOSTR3410_KEY_WRAP_PARAMS CK_GOSTR3410_KEY_WRAP_PARAMS
- typedef CK_GOSTR3410_KEY_WRAP_PARAMS CK_PTR CK_GOSTR3410_KEY_WRAP_PARAMS_PTR
- typedef struct CK_SEED_CBC_ENCRYPT_DATA_PARAMS CK_SEED_CBC_ENCRYPT_DATA_PARAMS
- typedef CK_SEED_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_SEED_CBC_ENCRYPT_DATA_PARAMS_PTR

Functions

- typedef CK_CALLBACK_FUNCTION (CK_RV, CK_NOTIFY)(CK_SESSION_HANDLE hSession
- typedef CK_CALLBACK_FUNCTION (CK_RV, CK_CREATEMUTEX)(CK_VOID_PTR_PTR ppMutex)
- typedef CK_CALLBACK_FUNCTION (CK_RV, CK_DESTROYMUTEX)(CK_VOID_PTR pMutex)
- typedef CK_CALLBACK_FUNCTION (CK_RV, CK_LOCKMUTEX)(CK_VOID_PTR pMutex)
- typedef CK_CALLBACK_FUNCTION (CK_RV, CK_UNLOCKMUTEX)(CK_VOID_PTR pMutex)

10.177.1 Macro Definition Documentation

10.177.1.1 CK_CERTIFICATE_CATEGORY_AUTHORITY

```
#define CK_CERTIFICATE_CATEGORY_AUTHORITY 2UL
```

10.177.1.2 CK_CERTIFICATE_CATEGORY_OTHER_ENTITY

```
#define CK_CERTIFICATE_CATEGORY_OTHER_ENTITY 3UL
```

10.177.1.3 CK_CERTIFICATE_CATEGORY_TOKEN_USER

```
#define CK_CERTIFICATE_CATEGORY_TOKEN_USER 1UL
```

10.177.1.4 CK_CERTIFICATE_CATEGORY_UNSPECIFIED

```
#define CK_CERTIFICATE_CATEGORY_UNSPECIFIED 0UL
```

10.177.1.5 CK_EFFECTIVELY_INFINITE

```
#define CK_EFFECTIVELY_INFINITE 0UL
```

10.177.1.6 CK_FALSE

```
#define CK_FALSE 0
```

10.177.1.7 CK_INVALID_HANDLE

```
#define CK_INVALID_HANDLE 0UL
```

10.177.1.8 CK_OTP_CHALLENGE

```
#define CK_OTP_CHALLENGE 2UL
```

10.177.1.9 CK_OTP_COUNTER

```
#define CK_OTP_COUNTER 4UL
```

10.177.1.10 CK_OTP_FLAGS

```
#define CK_OTP_FLAGS 5UL
```

10.177.1.11 CK_OTP_FORMAT_ALPHANUMERIC

```
#define CK_OTP_FORMAT_ALPHANUMERIC 2UL
```

10.177.1.12 CK_OTP_FORMAT_BINARY

```
#define CK_OTP_FORMAT_BINARY 3UL
```

10.177.1.13 CK_OTP_FORMAT_DECIMAL

```
#define CK_OTP_FORMAT_DECIMAL 0UL
```

10.177.1.14 CK_OTP_FORMAT_HEXADECIMAL

```
#define CK_OTP_FORMAT_HEXADECIMAL 1UL
```

10.177.1.15 CK_OTP_OUTPUT_FORMAT

```
#define CK_OTP_OUTPUT_FORMAT 7UL
```

10.177.1.16 CK_OTP_OUTPUT_LENGTH

```
#define CK_OTP_OUTPUT_LENGTH 6UL
```

10.177.1.17 CK_OTP_PARAM_IGNORED

```
#define CK_OTP_PARAM_IGNORED 0UL
```

10.177.1.18 CK_OTP_PARAM_MANDATORY

```
#define CK_OTP_PARAM_MANDATORY 2UL
```

10.177.1.19 CK_OTP_PARAM_OPTIONAL

```
#define CK_OTP_PARAM_OPTIONAL 1UL
```

10.177.1.20 CK_OTP_PIN

```
#define CK_OTP_PIN 1UL
```

10.177.1.21 CK_OTP_TIME

```
#define CK_OTP_TIME 3UL
```

10.177.1.22 CK_OTP_VALUE

```
#define CK_OTP_VALUE 0UL
```

10.177.1.23 CK_SECURITY_DOMAIN_MANUFACTURER

```
#define CK_SECURITY_DOMAIN_MANUFACTURER 1UL
```

10.177.1.24 CK_SECURITY_DOMAIN_OPERATOR

```
#define CK_SECURITY_DOMAIN_OPERATOR 2UL
```

10.177.1.25 CK_SECURITY_DOMAIN_THIRD_PARTY

```
#define CK_SECURITY_DOMAIN_THIRD_PARTY 3UL
```

10.177.1.26 CK_SECURITY_DOMAIN_UNSPECIFIED

```
#define CK_SECURITY_DOMAIN_UNSPECIFIED 0UL
```

10.177.1.27 CK_TRUE

```
#define CK_TRUE 1
```

10.177.1.28 CK_UNAVAILABLE_INFORMATION

```
#define CK_UNAVAILABLE_INFORMATION (~0UL)
```

10.177.1.29 CKA_AC_ISSUER

```
#define CKA_AC_ISSUER 0x00000083UL
```

10.177.1.30 CKA_ALLOWED_MECHANISMS

```
#define CKA_ALLOWED_MECHANISMS (CKF_ARRAY_ATTRIBUTE | 0x00000600UL)
```

10.177.1.31 CKA_ALWAYS_AUTHENTICATE

```
#define CKA_ALWAYS_AUTHENTICATE 0x00000202UL
```

10.177.1.32 CKA_ALWAYS_SENSITIVE

```
#define CKA_ALWAYS_SENSITIVE 0x00000165UL
```

10.177.1.33 CKA_APPLICATION

```
#define CKA_APPLICATION 0x00000010UL
```

10.177.1.34 CKA_ATTR_TYPES

```
#define CKA_ATTR_TYPES 0x00000085UL
```

10.177.1.35 CKA_AUTH_PIN_FLAGS

```
#define CKA_AUTH_PIN_FLAGS 0x00000201UL /* Deprecated */
```

10.177.1.36 CKA_BASE

```
#define CKA_BASE 0x00000132UL
```

10.177.1.37 CKA_BITS_PER_PIXEL

```
#define CKA_BITS_PER_PIXEL 0x00000406UL
```

10.177.1.38 CKA_CERTIFICATE_CATEGORY

```
#define CKA_CERTIFICATE_CATEGORY 0x00000087UL
```

10.177.1.39 CKA_CERTIFICATE_TYPE

```
#define CKA_CERTIFICATE_TYPE 0x00000080UL
```

10.177.1.40 CKA_CHAR_COLUMNS

```
#define CKA_CHAR_COLUMNS 0x00000404UL
```

10.177.1.41 CKA_CHAR_ROWS

```
#define CKA_CHAR_ROWS 0x00000403UL
```

10.177.1.42 CKA_CHAR_SETS

```
#define CKA_CHAR_SETS 0x00000480UL
```

10.177.1.43 CKA_CHECK_VALUE

```
#define CKA_CHECK_VALUE 0x00000090UL
```

10.177.1.44 CKA_CLASS

```
#define CKA_CLASS 0x00000000UL
```

10.177.1.45 CKA_COEFFICIENT

```
#define CKA_COEFFICIENT 0x00000128UL
```

10.177.1.46 CKA_COLOR

```
#define CKA_COLOR 0x00000405UL
```

10.177.1.47 CKA_COPYABLE

```
#define CKA_COPYABLE 0x00000171UL
```

10.177.1.48 CKA_DECRYPT

```
#define CKA_DECRYPT 0x00000105UL
```

10.177.1.49 CKA_DEFAULT_CMS_ATTRIBUTES

```
#define CKA_DEFAULT_CMS_ATTRIBUTES 0x00000502UL
```

10.177.1.50 CKA_DERIVE

```
#define CKA_DERIVE 0x0000010CUL
```

10.177.1.51 CKA_DERIVE_TEMPLATE

```
#define CKA_DERIVE_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000213UL)
```

10.177.1.52 CKA_DESTROYABLE

```
#define CKA_DESTROYABLE 0x00000172UL
```

10.177.1.53 CKA_EC_PARAMS

```
#define CKA_EC_PARAMS 0x00000180UL
```

10.177.1.54 CKA_EC_POINT

```
#define CKA_EC_POINT 0x00000181UL
```

10.177.1.55 CKA_ECDSA_PARAMS

```
#define CKA_ECDSA_PARAMS 0x00000180UL /* Deprecated */
```

10.177.1.56 CKA_ENCODING_METHODS

```
#define CKA_ENCODING_METHODS 0x00000481UL
```

10.177.1.57 CKA_ENCRYPT

```
#define CKA_ENCRYPT 0x00000104UL
```

10.177.1.58 CKA_END_DATE

```
#define CKA_END_DATE 0x00000111UL
```

10.177.1.59 CKA_EXPONENT_1

```
#define CKA_EXPONENT_1 0x00000126UL
```

10.177.1.60 CKA_EXPONENT_2

```
#define CKA_EXPONENT_2 0x00000127UL
```

10.177.1.61 CKA_EXTRACTABLE

```
#define CKA_EXTRACTABLE 0x00000162UL
```

10.177.1.62 CKA_GOST28147_PARAMS

```
#define CKA_GOST28147_PARAMS 0x00000252UL
```

10.177.1.63 CKA_GOSTR3410_PARAMS

```
#define CKA_GOSTR3410_PARAMS 0x00000250UL
```

10.177.1.64 CKA_GOSTR3411_PARAMS

```
#define CKA_GOSTR3411_PARAMS 0x00000251UL
```

10.177.1.65 CKA_HAS_RESET

```
#define CKA_HAS_RESET 0x00000302UL
```

10.177.1.66 CKA_HASH_OF_ISSUER_PUBLIC_KEY

```
#define CKA_HASH_OF_ISSUER_PUBLIC_KEY 0x0000008BUL
```

10.177.1.67 CKA_HASH_OF_SUBJECT_PUBLIC_KEY

```
#define CKA_HASH_OF_SUBJECT_PUBLIC_KEY 0x0000008AUL
```

10.177.1.68 CKA_HW_FEATURE_TYPE

```
#define CKA_HW_FEATURE_TYPE 0x00000300UL
```

10.177.1.69 CKA_ID

```
#define CKA_ID 0x00000102UL
```

10.177.1.70 CKA_ISSUER

```
#define CKA_ISSUER 0x00000081UL
```

10.177.1.71 CKA_JAVA_MIDP_SECURITY_DOMAIN

```
#define CKA_JAVA_MIDP_SECURITY_DOMAIN 0x00000088UL
```

10.177.1.72 CKA_KEY_GEN_MECHANISM

```
#define CKA_KEY_GEN_MECHANISM 0x00000166UL
```

10.177.1.73 CKA_KEY_TYPE

```
#define CKA_KEY_TYPE 0x00000100UL
```

10.177.1.74 CKA_LABEL

```
#define CKA_LABEL 0x00000003UL
```

10.177.1.75 CKA_LOCAL

```
#define CKA_LOCAL 0x00000163UL
```

10.177.1.76 CKA_MECHANISM_TYPE

```
#define CKA_MECHANISM_TYPE 0x00000500UL
```

10.177.1.77 CKA_MIME_TYPES

```
#define CKA_MIME_TYPES 0x00000482UL
```

10.177.1.78 CKA_MODIFIABLE

```
#define CKA_MODIFIABLE 0x00000170UL
```

10.177.1.79 CKA_MODULUS

```
#define CKA_MODULUS 0x00000120UL
```

10.177.1.80 CKA_MODULUS_BITS

```
#define CKA_MODULUS_BITS 0x00000121UL
```

10.177.1.81 CKA_NAME_HASH_ALGORITHM

```
#define CKA_NAME_HASH_ALGORITHM 0x0000008CUL
```

10.177.1.82 CKA_NEVER_EXTRACTABLE

```
#define CKA_NEVER_EXTRACTABLE 0x00000164UL
```

10.177.1.83 CKA_OBJECT_ID

```
#define CKA_OBJECT_ID 0x00000012UL
```

10.177.1.84 CKA_OTP_CHALLENGE_REQUIREMENT

```
#define CKA_OTP_CHALLENGE_REQUIREMENT 0x00000224UL
```

10.177.1.85 CKA_OTP_COUNTER

```
#define CKA_OTP_COUNTER 0x0000022EUL
```

10.177.1.86 CKA_OTP_COUNTER_REQUIREMENT

```
#define CKA_OTP_COUNTER_REQUIREMENT 0x00000226UL
```

10.177.1.87 CKA_OTP_FORMAT

```
#define CKA_OTP_FORMAT 0x00000220UL
```

10.177.1.88 CKA_OTP_LENGTH

```
#define CKA_OTP_LENGTH 0x00000221UL
```

10.177.1.89 CKA_OTP_PIN_REQUIREMENT

```
#define CKA_OTP_PIN_REQUIREMENT 0x00000227UL
```

10.177.1.90 CKA_OTP_SERVICE_IDENTIFIER

```
#define CKA_OTP_SERVICE_IDENTIFIER 0x0000022BUL
```

10.177.1.91 CKA_OTP_SERVICE_LOGO

```
#define CKA_OTP_SERVICE_LOGO 0x0000022CUL
```

10.177.1.92 CKA_OTP_SERVICE_LOGO_TYPE

```
#define CKA_OTP_SERVICE_LOGO_TYPE 0x0000022DUL
```

10.177.1.93 CKA_OTP_TIME

```
#define CKA_OTP_TIME 0x0000022FUL
```

10.177.1.94 CKA_OTP_TIME_INTERVAL

```
#define CKA_OTP_TIME_INTERVAL 0x00000222UL
```

10.177.1.95 CKA_OTP_TIME_REQUIREMENT

```
#define CKA_OTP_TIME_REQUIREMENT 0x00000225UL
```

10.177.1.96 CKA_OTP_USER_FRIENDLY_MODE

```
#define CKA_OTP_USER_FRIENDLY_MODE 0x00000223UL
```

10.177.1.97 CKA_OTP_USER_IDENTIFIER

```
#define CKA_OTP_USER_IDENTIFIER 0x0000022AUL
```

10.177.1.98 CKA_OWNER

```
#define CKA_OWNER 0x00000084UL
```

10.177.1.99 CKA_PIXEL_X

```
#define CKA_PIXEL_X 0x00000400UL
```

10.177.1.100 CKA_PIXEL_Y

```
#define CKA_PIXEL_Y 0x00000401UL
```

10.177.1.101 CKA_PRIME

```
#define CKA_PRIME 0x00000130UL
```

10.177.1.102 CKA_PRIME_1

```
#define CKA_PRIME_1 0x00000124UL
```

10.177.1.103 CKA_PRIME_2

```
#define CKA_PRIME_2 0x00000125UL
```

10.177.1.104 CKA_PRIME_BITS

```
#define CKA_PRIME_BITS 0x00000133UL
```

10.177.1.105 CKA_PRIVATE

```
#define CKA_PRIVATE 0x00000002UL
```

10.177.1.106 CKA_PRIVATE_EXPONENT

```
#define CKA_PRIVATE_EXPONENT 0x00000123UL
```

10.177.1.107 CKA_PUBLIC_EXPONENT

```
#define CKA_PUBLIC_EXPONENT 0x00000122UL
```

10.177.1.108 CKA_PUBLIC_KEY_INFO

```
#define CKA_PUBLIC_KEY_INFO 0x00000129UL
```

10.177.1.109 CKA_REQUIRED_CMS_ATTRIBUTES

```
#define CKA_REQUIRED_CMS_ATTRIBUTES 0x00000501UL
```

10.177.1.110 CKA_RESET_ON_INIT

```
#define CKA_RESET_ON_INIT 0x00000301UL
```

10.177.1.111 CKA_RESOLUTION

```
#define CKA_RESOLUTION 0x00000402UL
```

10.177.1.112 CKA_SECONDARY_AUTH

```
#define CKA_SECONDARY_AUTH 0x00000200UL /* Deprecated */
```

10.177.1.113 CKA_SENSITIVE

```
#define CKA_SENSITIVE 0x00000103UL
```

10.177.1.114 CKA_SERIAL_NUMBER

```
#define CKA_SERIAL_NUMBER 0x00000082UL
```

10.177.1.115 CKA_SIGN

```
#define CKA_SIGN 0x00000108UL
```

10.177.1.116 CKA_SIGN_RECOVER

```
#define CKA_SIGN_RECOVER 0x00000109UL
```

10.177.1.117 CKA_START_DATE

```
#define CKA_START_DATE 0x00000110UL
```

10.177.1.118 CKA_SUB_PRIME_BITS

```
#define CKA_SUB_PRIME_BITS CKA_SUBPRIME_BITS
```

10.177.1.119 CKA_SUBJECT

```
#define CKA_SUBJECT 0x00000101UL
```

10.177.1.120 CKA_SUBPRIME

```
#define CKA_SUBPRIME 0x00000131UL
```

10.177.1.121 CKA_SUBPRIME_BITS

```
#define CKA_SUBPRIME_BITS 0x00000134UL
```

10.177.1.122 CKA_SUPPORTED_CMS_ATTRIBUTES

```
#define CKA_SUPPORTED_CMS_ATTRIBUTES 0x00000503UL
```

10.177.1.123 CKA_TOKEN

```
#define CKA_TOKEN 0x00000001UL
```

10.177.1.124 CKA_TRUSTED

```
#define CKA_TRUSTED 0x00000086UL
```

10.177.1.125 CKA_UNWRAP

```
#define CKA_UNWRAP 0x00000107UL
```

10.177.1.126 CKA_UNWRAP_TEMPLATE

```
#define CKA_UNWRAP_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000212UL)
```

10.177.1.127 CKA_URL

```
#define CKA_URL 0x00000089UL
```

10.177.1.128 CKA_VALUE

```
#define CKA_VALUE 0x00000011UL
```

10.177.1.129 CKA_VALUE_BITS

```
#define CKA_VALUE_BITS 0x00000160UL
```

10.177.1.130 CKA_VALUE_LEN

```
#define CKA_VALUE_LEN 0x00000161UL
```

10.177.1.131 CKA_VENDOR_DEFINED

```
#define CKA_VENDOR_DEFINED 0x80000000UL
```

10.177.1.132 CKA_VERIFY

```
#define CKA_VERIFY 0x0000010AUL
```

10.177.1.133 CKA_VERIFY_RECOVER

```
#define CKA_VERIFY_RECOVER 0x0000010BUL
```

10.177.1.134 CKA_WRAP

```
#define CKA_WRAP 0x00000106UL
```

10.177.1.135 CKA_WRAP_TEMPLATE

```
#define CKA_WRAP_TEMPLATE (CKF_ARRAY_ATTRIBUTE | 0x00000211UL)
```

10.177.1.136 CKA_WRAP_WITH_TRUSTED

```
#define CKA_WRAP_WITH_TRUSTED 0x00000210UL
```

10.177.1.137 CKC_OPENPGP

```
#define CKC_OPENPGP (CKC_VENDOR_DEFINED | 0x00504750)
```

10.177.1.138 CKC_VENDOR_DEFINED

```
#define CKC_VENDOR_DEFINED 0x80000000UL
```

10.177.1.139 CKC_WTLS

```
#define CKC_WTLS 0x00000002UL
```

10.177.1.140 CKC_X_509

```
#define CKC_X_509 0x00000000UL
```

10.177.1.141 CKC_X_509_ATTR_CERT

```
#define CKC_X_509_ATTR_CERT 0x00000001UL
```

10.177.1.142 CKD_CPDIVERSIFY_KDF

```
#define CKD_CPDIVERSIFY_KDF 0x00000009UL
```

10.177.1.143 CKD_NULL

```
#define CKD_NULL 0x00000001UL
```

10.177.1.144 CKD_SHA1_KDF

```
#define CKD_SHA1_KDF 0x00000002UL
```

10.177.1.145 CKD_SHA1_KDF_ASN1

```
#define CKD_SHA1_KDF_ASN1 0x00000003UL
```

10.177.1.146 CKD_SHA1_KDF_CONCATENATE

```
#define CKD_SHA1_KDF_CONCATENATE 0x00000004UL
```

10.177.1.147 CKD_SHA224_KDF

```
#define CKD_SHA224_KDF 0x00000005UL
```

10.177.1.148 CKD_SHA256_KDF

```
#define CKD_SHA256_KDF 0x00000006UL
```

10.177.1.149 CKD_SHA384_KDF

```
#define CKD_SHA384_KDF 0x00000007UL
```

10.177.1.150 CKD_SHA512_KDF

```
#define CKD_SHA512_KDF 0x00000008UL
```

10.177.1.151 CKF_ARRAY_ATTRIBUTE

```
#define CKF_ARRAY_ATTRIBUTE 0x40000000UL
```

10.177.1.152 CKF_CLOCK_ON_TOKEN

```
#define CKF_CLOCK_ON_TOKEN 0x00000040UL
```

10.177.1.153 CKF_DECRYPT

```
#define CKF_DECRYPT 0x00000200UL
```

10.177.1.154 CKF_DERIVE

```
#define CKF_DERIVE 0x00080000UL
```

10.177.1.155 CKF_DIGEST

```
#define CKF_DIGEST 0x00000400UL
```

10.177.1.156 CKF_DONT_BLOCK

```
#define CKF_DONT_BLOCK 1
```

10.177.1.157 CKF_DUAL_CRYPTO_OPERATIONS

```
#define CKF_DUAL_CRYPTO_OPERATIONS 0x00000200UL
```

10.177.1.158 CKF_EC_COMPRESS

```
#define CKF_EC_COMPRESS 0x02000000UL
```

10.177.1.159 CKF_EC_ECPARAMETERS

```
#define CKF_EC_ECPARAMETERS 0x00400000UL
```

10.177.1.160 CKF_EC_F_2M

```
#define CKF_EC_F_2M 0x00200000UL
```

10.177.1.161 CKF_EC_F_P

```
#define CKF_EC_F_P 0x00100000UL
```

10.177.1.162 CKF_EC_NAMEDCURVE

```
#define CKF_EC_NAMEDCURVE 0x00800000UL
```

10.177.1.163 CKF_EC_UNCOMPRESS

```
#define CKF_EC_UNCOMPRESS 0x01000000UL
```

10.177.1.164 CKF_ENCRYPT

```
#define CKF_ENCRYPT 0x00000100UL
```

10.177.1.165 CKF_ERROR_STATE

```
#define CKF_ERROR_STATE 0x01000000UL
```

10.177.1.166 CKF_EXCLUDE_CHALLENGE

```
#define CKF_EXCLUDE_CHALLENGE 0x00000008UL
```

10.177.1.167 CKF_EXCLUDE_COUNTER

```
#define CKF_EXCLUDE_COUNTER 0x00000004UL
```

10.177.1.168 CKF_EXCLUDE_PIN

```
#define CKF_EXCLUDE_PIN 0x00000010UL
```

10.177.1.169 CKF_EXCLUDE_TIME

```
#define CKF_EXCLUDE_TIME 0x00000002UL
```

10.177.1.170 CKF_EXTENSION

```
#define CKF_EXTENSION 0x80000000UL
```

10.177.1.171 CKF_GENERATE

```
#define CKF_GENERATE 0x00008000UL
```

10.177.1.172 CKF_GENERATE_KEY_PAIR

```
#define CKF_GENERATE_KEY_PAIR 0x00010000UL
```

10.177.1.173 CKF_HW

```
#define CKF_HW 0x00000001UL /* performed by HW */
```

10.177.1.174 CKF_HW_SLOT

```
#define CKF_HW_SLOT 0x00000004UL /* hardware slot */
```

10.177.1.175 CKF_LIBRARY_CANT_CREATE_OS_THREADS

```
#define CKF_LIBRARY_CANT_CREATE_OS_THREADS 0x00000001UL
```

10.177.1.176 CKF_LOGIN_REQUIRED

```
#define CKF_LOGIN_REQUIRED 0x00000004UL /* user must login */
```

10.177.1.177 CKF_NEXT_OTP

```
#define CKF_NEXT_OTP 0x00000001UL
```

10.177.1.178 CKF_OS_LOCKING_OK

```
#define CKF_OS_LOCKING_OK 0x00000002UL
```

10.177.1.179 CKF_PROTECTED_AUTHENTICATION_PATH

```
#define CKF_PROTECTED_AUTHENTICATION_PATH 0x00000100UL
```

10.177.1.180 CKF_REMOVABLE_DEVICE

```
#define CKF_REMOVABLE_DEVICE 0x00000002UL /* removable devices*/
```

10.177.1.181 CKF_RESTORE_KEY_NOT_NEEDED

```
#define CKF_RESTORE_KEY_NOT_NEEDED 0x00000020UL
```

10.177.1.182 CKF_RNG

```
#define CKF_RNG 0x00000001UL /* has random # generator */
```

10.177.1.183 CKF_RW_SESSION

```
#define CKF_RW_SESSION 0x00000002UL /* session is r/w */
```

10.177.1.184 CKF_SECONDARY_AUTHENTICATION

```
#define CKF_SECONDARY_AUTHENTICATION 0x00000800UL
```

10.177.1.185 CKF_SERIAL_SESSION

```
#define CKF_SERIAL_SESSION 0x00000004UL /* no parallel */
```

10.177.1.186 CKF_SIGN

```
#define CKF_SIGN 0x00000800UL
```

10.177.1.187 CKF_SIGN_RECOVER

```
#define CKF_SIGN_RECOVER 0x00001000UL
```

10.177.1.188 CKF_SO_PIN_COUNT_LOW

```
#define CKF_SO_PIN_COUNT_LOW 0x00100000UL
```

10.177.1.189 CKF_SO_PIN_FINAL_TRY

```
#define CKF_SO_PIN_FINAL_TRY 0x00200000UL
```

10.177.1.190 CKF_SO_PIN_LOCKED

```
#define CKF_SO_PIN_LOCKED 0x00400000UL
```

10.177.1.191 CKF_SO_PIN_TO_BE_CHANGED

```
#define CKF_SO_PIN_TO_BE_CHANGED 0x00800000UL
```

10.177.1.192 CKF_TOKEN_INITIALIZED

```
#define CKF_TOKEN_INITIALIZED 0x00000400UL
```

10.177.1.193 CKF_TOKEN_PRESENT

```
#define CKF_TOKEN_PRESENT 0x00000001UL /* a token is there */
```

10.177.1.194 CKF_UNWRAP

```
#define CKF_UNWRAP 0x00040000UL
```

10.177.1.195 CKF_USER_FRIENDLY_OTP

```
#define CKF_USER_FRIENDLY_OTP 0x00000020UL
```

10.177.1.196 CKF_USER_PIN_COUNT_LOW

```
#define CKF_USER_PIN_COUNT_LOW 0x00010000UL
```

10.177.1.197 CKF_USER_PIN_FINAL_TRY

```
#define CKF_USER_PIN_FINAL_TRY 0x00020000UL
```

10.177.1.198 CKF_USER_PIN_INITIALIZED

```
#define CKF_USER_PIN_INITIALIZED 0x00000008UL /* normal user's PIN is set */
```

10.177.1.199 CKF_USER_PIN_LOCKED

```
#define CKF_USER_PIN_LOCKED 0x00040000UL
```

10.177.1.200 CKF_USER_PIN_TO_BE_CHANGED

```
#define CKF_USER_PIN_TO_BE_CHANGED 0x00080000UL
```

10.177.1.201 CKF_VERIFY

```
#define CKF_VERIFY 0x00002000UL
```

10.177.1.202 CKF_VERIFY_RECOVER

```
#define CKF_VERIFY_RECOVER 0x00004000UL
```

10.177.1.203 CKF_WRAP

```
#define CKF_WRAP 0x00020000UL
```

10.177.1.204 CKF_WRITE_PROTECTED

```
#define CKF_WRITE_PROTECTED 0x00000002UL /* token is write-protected */
```

10.177.1.205 CKG_MGF1_SHA1

```
#define CKG_MGF1_SHA1 0x00000001UL
```

10.177.1.206 CKG_MGF1_SHA224

```
#define CKG_MGF1_SHA224 0x00000005UL
```

10.177.1.207 CKG_MGF1_SHA256

```
#define CKG_MGF1_SHA256 0x00000002UL
```

10.177.1.208 CKG_MGF1_SHA384

```
#define CKG_MGF1_SHA384 0x00000003UL
```

10.177.1.209 CKG_MGF1_SHA512

```
#define CKG_MGF1_SHA512 0x00000004UL
```

10.177.1.210 CKH_CLOCK

```
#define CKH_CLOCK 0x00000002UL
```

10.177.1.211 CKH_MONOTONIC_COUNTER

```
#define CKH_MONOTONIC_COUNTER 0x00000001UL
```

10.177.1.212 CKH_USER_INTERFACE

```
#define CKH_USER_INTERFACE 0x00000003UL
```

10.177.1.213 CKH_VENDOR_DEFINED

```
#define CKH_VENDOR_DEFINED 0x80000000UL
```

10.177.1.214 CKK_ACTI

```
#define CKK_ACTI 0x00000024UL
```

10.177.1.215 CKK_AES

```
#define CKK_AES 0x0000001FUL
```

10.177.1.216 CKK_ARIA

```
#define CKK_ARIA 0x00000026UL
```

10.177.1.217 CKK_BATON

```
#define CKK_BATON 0x0000001CUL
```

10.177.1.218 CKK_BLOWFISH

```
#define CKK_BLOWFISH 0x00000020UL
```

10.177.1.219 CKK_CAMELLIA

```
#define CKK_CAMELLIA 0x00000025UL
```

10.177.1.220 CKK_CAST

```
#define CKK_CAST 0x00000016UL
```

10.177.1.221 CKK_CAST128

```
#define CKK_CAST128 0x00000018UL
```

10.177.1.222 CKK_CAST3

```
#define CKK_CAST3 0x00000017UL
```

10.177.1.223 CKK_CAST5

```
#define CKK_CAST5 0x00000018UL /* Deprecated */
```

10.177.1.224 CKK_CDMF

```
#define CKK_CDMF 0x0000001EUL
```

10.177.1.225 CKK_DES

```
#define CKK_DES 0x00000013UL
```

10.177.1.226 CKK_DES2

```
#define CKK_DES2 0x00000014UL
```

10.177.1.227 CKK_DES3

```
#define CKK_DES3 0x00000015UL
```

10.177.1.228 CKK_DH

```
#define CKK_DH 0x00000002UL
```

10.177.1.229 CKK_DSA

```
#define CKK_DSA 0x00000001UL
```

10.177.1.230 CKK_EC

```
#define CKK_EC 0x00000003UL
```

10.177.1.231 CKK_ECDSA

```
#define CKK_ECDSA 0x00000003UL /* Deprecated */
```

10.177.1.232 CKK_GENERIC_SECRET

```
#define CKK_GENERIC_SECRET 0x00000010UL
```

10.177.1.233 CKK_GOST28147

```
#define CKK_GOST28147 0x00000032UL
```

10.177.1.234 CKK_GOSTR3410

```
#define CKK_GOSTR3410 0x00000030UL
```

10.177.1.235 CKK_GOSTR3411

```
#define CKK_GOSTR3411 0x00000031UL
```

10.177.1.236 CKK_HOTP

```
#define CKK_HOTP 0x00000023UL
```

10.177.1.237 CKK_IDEA

```
#define CKK_IDEA 0x0000001AUL
```

10.177.1.238 CKK_JUNIPER

```
#define CKK_JUNIPER 0x0000001DUL
```

10.177.1.239 CKK_KEA

```
#define CKK_KEA 0x00000005UL
```

10.177.1.240 CKK_MD5_HMAC

```
#define CKK_MD5_HMAC 0x00000027UL
```

10.177.1.241 CKK_RC2

```
#define CKK_RC2 0x00000011UL
```

10.177.1.242 CKK_RC4

```
#define CKK_RC4 0x00000012UL
```

10.177.1.243 CKK_RC5

```
#define CKK_RC5 0x00000019UL
```

10.177.1.244 CKK_RIPEMD128_HMAC

```
#define CKK_RIPEMD128_HMAC 0x00000029UL
```

10.177.1.245 CKK_RIPEMD160_HMAC

```
#define CKK_RIPEMD160_HMAC 0x0000002AUL
```

10.177.1.246 CKK_RSA

```
#define CKK_RSA 0x00000000UL
```

10.177.1.247 CKK_SECURID

```
#define CKK_SECURID 0x00000022UL
```

10.177.1.248 CKK_SEED

```
#define CKK_SEED 0x0000002FUL
```

10.177.1.249 CKK_SHA224_HMAC

```
#define CKK_SHA224_HMAC 0x0000002EUL
```

10.177.1.250 CKK_SHA256_HMAC

```
#define CKK_SHA256_HMAC 0x0000002BUL
```

10.177.1.251 CKK_SHA384_HMAC

```
#define CKK_SHA384_HMAC 0x0000002CUL
```

10.177.1.252 CKK_SHA512_HMAC

```
#define CKK_SHA512_HMAC 0x0000002DUL
```

10.177.1.253 CKK_SHA_1_HMAC

```
#define CKK_SHA_1_HMAC 0x00000028UL
```

10.177.1.254 CKK_SKIPJACK

```
#define CKK_SKIPJACK 0x0000001BUL
```

10.177.1.255 CKK_TWOFISH

```
#define CKK_TWOFISH 0x00000021UL
```

10.177.1.256 CKK_VENDOR_DEFINED

```
#define CKK_VENDOR_DEFINED 0x80000000UL
```

10.177.1.257 CKK_X9_42_DH

```
#define CKK_X9_42_DH 0x00000004UL
```

10.177.1.258 CKM_ACTI

```
#define CKM_ACTI 0x000002A0UL
```

10.177.1.259 CKM_ACTI_KEY_GEN

```
#define CKM_ACTI_KEY_GEN 0x000002A1UL
```

10.177.1.260 CKM_AES_CBC

```
#define CKM_AES_CBC 0x00001082UL
```

10.177.1.261 CKM_AES_CBC_ENCRYPT_DATA

```
#define CKM_AES_CBC_ENCRYPT_DATA 0x00001105UL
```

10.177.1.262 CKM_AES_CBC_PAD

```
#define CKM_AES_CBC_PAD 0x00001085UL
```

10.177.1.263 CKM_AES_CCM

```
#define CKM_AES_CCM 0x00001088UL
```

10.177.1.264 CKM_AES_CFB1

```
#define CKM_AES_CFB1 0x00002108UL
```

10.177.1.265 CKM_AES_CFB128

```
#define CKM_AES_CFB128 0x00002107UL
```

10.177.1.266 CKM_AES_CFB64

```
#define CKM_AES_CFB64 0x00002105UL
```

10.177.1.267 CKM_AES_CFB8

```
#define CKM_AES_CFB8 0x00002106UL
```

10.177.1.268 CKM_AES_CMAC

```
#define CKM_AES_CMAC 0x0000108AUL
```

10.177.1.269 CKM_AES_CMAC_GENERAL

```
#define CKM_AES_CMAC_GENERAL 0x0000108BUL
```

10.177.1.270 CKM_AES_CTR

```
#define CKM_AES_CTR 0x00001086UL
```

10.177.1.271 CKM_AES_CTS

```
#define CKM_AES_CTS 0x00001089UL
```

10.177.1.272 CKM_AES_ECB

```
#define CKM_AES_ECB 0x00001081UL
```

10.177.1.273 CKM_AES_ECB_ENCRYPT_DATA

```
#define CKM_AES_ECB_ENCRYPT_DATA 0x00001104UL
```

10.177.1.274 CKM_AES_GCM

```
#define CKM_AES_GCM 0x00001087UL
```

10.177.1.275 CKM_AES_GMAC

```
#define CKM_AES_GMAC 0x0000108EUL
```

10.177.1.276 CKM_AES_KEY_GEN

```
#define CKM_AES_KEY_GEN 0x00001080UL
```

10.177.1.277 CKM_AES_KEY_WRAP

```
#define CKM_AES_KEY_WRAP 0x00002109UL /* WAS: 0x00001090 */
```

10.177.1.278 CKM_AES_KEY_WRAP_PAD

```
#define CKM_AES_KEY_WRAP_PAD 0x0000210AUL /* WAS: 0x00001091 */
```

10.177.1.279 CKM_AES_MAC

```
#define CKM_AES_MAC 0x00001083UL
```

10.177.1.280 CKM_AES_MAC_GENERAL

```
#define CKM_AES_MAC_GENERAL 0x00001084UL
```

10.177.1.281 CKM_AES_OFB

```
#define CKM_AES_OFB 0x00002104UL
```

10.177.1.282 CKM_AES_XCBC_MAC

```
#define CKM_AES_XCBC_MAC 0x0000108CUL
```

10.177.1.283 CKM_AES_XCBC_MAC_96

```
#define CKM_AES_XCBC_MAC_96 0x0000108DUL
```

10.177.1.284 CKM_ARIA_CBC

```
#define CKM_ARIA_CBC 0x00000562UL
```

10.177.1.285 CKM_ARIA_CBC_ENCRYPT_DATA

```
#define CKM_ARIA_CBC_ENCRYPT_DATA 0x00000567UL
```

10.177.1.286 CKM_ARIA_CBC_PAD

```
#define CKM_ARIA_CBC_PAD 0x00000565UL
```

10.177.1.287 CKM_ARIA_ECB

```
#define CKM_ARIA_ECB 0x00000561UL
```

10.177.1.288 CKM_ARIA_ECB_ENCRYPT_DATA

```
#define CKM_ARIA_ECB_ENCRYPT_DATA 0x00000566UL
```

10.177.1.289 CKM_ARIA_KEY_GEN

```
#define CKM_ARIA_KEY_GEN 0x00000560UL
```

10.177.1.290 CKM_ARIA_MAC

```
#define CKM_ARIA_MAC 0x00000563UL
```

10.177.1.291 CKM_ARIA_MAC_GENERAL

```
#define CKM_ARIA_MAC_GENERAL 0x00000564UL
```

10.177.1.292 CKM_BATON_CBC128

```
#define CKM_BATON_CBC128 0x00001033UL
```

10.177.1.293 CKM_BATON_COUNTER

```
#define CKM_BATON_COUNTER 0x00001034UL
```

10.177.1.294 CKM_BATON_ECB128

```
#define CKM_BATON_ECB128 0x00001031UL
```

10.177.1.295 CKM_BATON_ECB96

```
#define CKM_BATON_ECB96 0x00001032UL
```

10.177.1.296 CKM_BATON_KEY_GEN

```
#define CKM_BATON_KEY_GEN 0x00001030UL
```

10.177.1.297 CKM_BATON_SHUFFLE

```
#define CKM_BATON_SHUFFLE 0x00001035UL
```

10.177.1.298 CKM_BATON_WRAP

```
#define CKM_BATON_WRAP 0x00001036UL
```

10.177.1.299 CKM_BLOWFISH_CBC

```
#define CKM_BLOWFISH_CBC 0x00001091UL
```

10.177.1.300 CKM_BLOWFISH_CBC_PAD

```
#define CKM_BLOWFISH_CBC_PAD 0x00001094UL
```

10.177.1.301 CKM_BLOWFISH_KEY_GEN

```
#define CKM_BLOWFISH_KEY_GEN 0x00001090UL
```

10.177.1.302 CKM_CAMELLIA_CBC

```
#define CKM_CAMELLIA_CBC 0x00000552UL
```

10.177.1.303 CKM_CAMELLIA_CBC_ENCRYPT_DATA

```
#define CKM_CAMELLIA_CBC_ENCRYPT_DATA 0x00000557UL
```

10.177.1.304 CKM_CAMELLIA_CBC_PAD

```
#define CKM_CAMELLIA_CBC_PAD 0x00000555UL
```

10.177.1.305 CKM_CAMELLIA_CTR

```
#define CKM_CAMELLIA_CTR 0x00000558UL
```

10.177.1.306 CKM_CAMELLIA_ECB

```
#define CKM_CAMELLIA_ECB 0x00000551UL
```

10.177.1.307 CKM_CAMELLIA_ECB_ENCRYPT_DATA

```
#define CKM_CAMELLIA_ECB_ENCRYPT_DATA 0x00000556UL
```

10.177.1.308 CKM_CAMELLIA_KEY_GEN

```
#define CKM_CAMELLIA_KEY_GEN 0x00000550UL
```

10.177.1.309 CKM_CAMELLIA_MAC

```
#define CKM_CAMELLIA_MAC 0x00000553UL
```

10.177.1.310 CKM_CAMELLIA_MAC_GENERAL

```
#define CKM_CAMELLIA_MAC_GENERAL 0x00000554UL
```

10.177.1.311 CKM_CAST128_CBC

```
#define CKM_CAST128_CBC 0x00000322UL
```

10.177.1.312 CKM_CAST128_CBC_PAD

```
#define CKM_CAST128_CBC_PAD 0x00000325UL
```

10.177.1.313 CKM_CAST128_ECB

```
#define CKM_CAST128_ECB 0x00000321UL
```

10.177.1.314 CKM_CAST128_KEY_GEN

```
#define CKM_CAST128_KEY_GEN 0x00000320UL
```

10.177.1.315 CKM_CAST128_MAC

```
#define CKM_CAST128_MAC 0x00000323UL
```

10.177.1.316 CKM_CAST128_MAC_GENERAL

```
#define CKM_CAST128_MAC_GENERAL 0x00000324UL
```

10.177.1.317 CKM_CAST3_CBC

```
#define CKM_CAST3_CBC 0x00000312UL
```

10.177.1.318 CKM_CAST3_CBC_PAD

```
#define CKM_CAST3_CBC_PAD 0x00000315UL
```

10.177.1.319 CKM_CAST3_ECB

```
#define CKM_CAST3_ECB 0x00000311UL
```

10.177.1.320 CKM_CAST3_KEY_GEN

```
#define CKM_CAST3_KEY_GEN 0x00000310UL
```

10.177.1.321 CKM_CAST3_MAC

```
#define CKM_CAST3_MAC 0x00000313UL
```

10.177.1.322 CKM_CAST3_MAC_GENERAL

```
#define CKM_CAST3_MAC_GENERAL 0x00000314UL
```

10.177.1.323 CKM_CAST5_CBC

```
#define CKM_CAST5_CBC 0x00000322UL /* Deprecated */
```

10.177.1.324 CKM_CAST5_CBC_PAD

```
#define CKM_CAST5_CBC_PAD 0x00000325UL /* Deprecated */
```

10.177.1.325 CKM_CAST5_ECB

```
#define CKM_CAST5_ECB 0x00000321UL
```

10.177.1.326 CKM_CAST5_KEY_GEN

```
#define CKM_CAST5_KEY_GEN 0x00000320UL
```

10.177.1.327 CKM_CAST5_MAC

```
#define CKM_CAST5_MAC 0x00000323UL /* Deprecated */
```

10.177.1.328 CKM_CAST5_MAC_GENERAL

```
#define CKM_CAST5_MAC_GENERAL 0x00000324UL /* Deprecated */
```

10.177.1.329 CKM_CAST_CBC

```
#define CKM_CAST_CBC 0x00000302UL
```

10.177.1.330 CKM_CAST_CBC_PAD

```
#define CKM_CAST_CBC_PAD 0x00000305UL
```

10.177.1.331 CKM_CAST_ECB

```
#define CKM_CAST_ECB 0x00000301UL
```

10.177.1.332 CKM_CAST_KEY_GEN

```
#define CKM_CAST_KEY_GEN 0x00000300UL
```

10.177.1.333 CKM_CAST_MAC

```
#define CKM_CAST_MAC 0x00000303UL
```

10.177.1.334 CKM_CAST_MAC_GENERAL

```
#define CKM_CAST_MAC_GENERAL 0x00000304UL
```

10.177.1.335 CKM_CDMF_CBC

```
#define CKM_CDMF_CBC 0x00000142UL
```

10.177.1.336 CKM_CDMF_CBC_PAD

```
#define CKM_CDMF_CBC_PAD 0x00000145UL
```

10.177.1.337 CKM_CDMF_ECB

```
#define CKM_CDMF_ECB 0x00000141UL
```

10.177.1.338 CKM_CDMF_KEY_GEN

```
#define CKM_CDMF_KEY_GEN 0x00000140UL
```

10.177.1.339 CKM_CDMF_MAC

```
#define CKM_CDMF_MAC 0x00000143UL
```

10.177.1.340 CKM_CDMF_MAC_GENERAL

```
#define CKM_CDMF_MAC_GENERAL 0x00000144UL
```

10.177.1.341 CKM_CMS_SIG

```
#define CKM_CMS_SIG 0x00000500UL
```

10.177.1.342 CKM_CONCATENATE_BASE_AND_DATA

```
#define CKM_CONCATENATE_BASE_AND_DATA 0x00000362UL
```

10.177.1.343 CKM_CONCATENATE_BASE_AND_KEY

```
#define CKM_CONCATENATE_BASE_AND_KEY 0x00000360UL
```

10.177.1.344 CKM_CONCATENATE_DATA_AND_BASE

```
#define CKM_CONCATENATE_DATA_AND_BASE 0x00000363UL
```

10.177.1.345 CKM_DES2_KEY_GEN

```
#define CKM_DES2_KEY_GEN 0x00000130UL
```

10.177.1.346 CKM_DES3_CBC

```
#define CKM_DES3_CBC 0x00000133UL
```

10.177.1.347 CKM_DES3_CBC_ENCRYPT_DATA

```
#define CKM_DES3_CBC_ENCRYPT_DATA 0x00001103UL
```

10.177.1.348 CKM_DES3_CBC_PAD

```
#define CKM_DES3_CBC_PAD 0x00000136UL
```

10.177.1.349 CKM_DES3_CMAC

```
#define CKM_DES3_CMAC 0x00000138UL
```

10.177.1.350 CKM_DES3_CMAC_GENERAL

```
#define CKM_DES3_CMAC_GENERAL 0x00000137UL
```

10.177.1.351 CKM_DES3_ECB

```
#define CKM_DES3_ECB 0x00000132UL
```

10.177.1.352 CKM_DES3_ECB_ENCRYPT_DATA

```
#define CKM_DES3_ECB_ENCRYPT_DATA 0x00001102UL
```

10.177.1.353 CKM_DES3_KEY_GEN

```
#define CKM_DES3_KEY_GEN 0x00000131UL
```

10.177.1.354 CKM_DES3_MAC

```
#define CKM_DES3_MAC 0x00000134UL
```

10.177.1.355 CKM_DES3_MAC_GENERAL

```
#define CKM_DES3_MAC_GENERAL 0x00000135UL
```

10.177.1.356 CKM_DES_CBC

```
#define CKM_DES_CBC 0x00000122UL
```

10.177.1.357 CKM_DES_CBC_ENCRYPT_DATA

```
#define CKM_DES_CBC_ENCRYPT_DATA 0x00001101UL
```

10.177.1.358 CKM_DES_CBC_PAD

```
#define CKM_DES_CBC_PAD 0x00000125UL
```

10.177.1.359 CKM_DES_CFB64

```
#define CKM_DES_CFB64 0x00000152UL
```

10.177.1.360 CKM_DES_CFB8

```
#define CKM_DES_CFB8 0x00000153UL
```

10.177.1.361 CKM_DES_ECB

```
#define CKM_DES_ECB 0x00000121UL
```

10.177.1.362 CKM_DES_ECB_ENCRYPT_DATA

```
#define CKM_DES_ECB_ENCRYPT_DATA 0x00001100UL
```

10.177.1.363 CKM_DES_KEY_GEN

```
#define CKM_DES_KEY_GEN 0x00000120UL
```

10.177.1.364 CKM_DES_MAC

```
#define CKM_DES_MAC 0x00000123UL
```

10.177.1.365 CKM_DES_MAC_GENERAL

```
#define CKM_DES_MAC_GENERAL 0x00000124UL
```

10.177.1.366 CKM_DES_OFB64

```
#define CKM_DES_OFB64 0x00000150UL
```

10.177.1.367 CKM_DES_OFB8

```
#define CKM_DES_OFB8 0x00000151UL
```

10.177.1.368 CKM_DH_PKCS_DERIVE

```
#define CKM_DH_PKCS_DERIVE 0x00000021UL
```

10.177.1.369 CKM_DH_PKCS_KEY_PAIR_GEN

```
#define CKM_DH_PKCS_KEY_PAIR_GEN 0x00000020UL
```

10.177.1.370 CKM_DH_PKCS_PARAMETER_GEN

```
#define CKM_DH_PKCS_PARAMETER_GEN 0x00002001UL
```

10.177.1.371 CKM_DSA

```
#define CKM_DSA 0x00000011UL
```

10.177.1.372 CKM_DSA_KEY_PAIR_GEN

```
#define CKM_DSA_KEY_PAIR_GEN 0x00000010UL
```

10.177.1.373 CKM_DSA_PARAMETER_GEN

```
#define CKM_DSA_PARAMETER_GEN 0x00002000UL
```

10.177.1.374 CKM_DSA_PROBABLISTIC_PARAMETER_GEN

```
#define CKM_DSA_PROBABLISTIC_PARAMETER_GEN 0x00002003UL
```

10.177.1.375 CKM_DSA_SHA1

```
#define CKM_DSA_SHA1 0x00000012UL
```

10.177.1.376 CKM_DSA_SHA224

```
#define CKM_DSA_SHA224 0x00000013UL
```

10.177.1.377 CKM_DSA_SHA256

```
#define CKM_DSA_SHA256 0x00000014UL
```

10.177.1.378 CKM_DSA_SHA384

```
#define CKM_DSA_SHA384 0x00000015UL
```

10.177.1.379 CKM_DSA_SHA512

```
#define CKM_DSA_SHA512 0x00000016UL
```

10.177.1.380 CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN

```
#define CKM_DSA_SHAWE_TAYLOR_PARAMETER_GEN 0x00002004UL
```

10.177.1.381 CKM_EC_KEY_PAIR_GEN

```
#define CKM_EC_KEY_PAIR_GEN 0x00001040UL
```

10.177.1.382 CKM_ECDH1_COFACTOR_DERIVE

```
#define CKM_ECDH1_COFACTOR_DERIVE 0x00001051UL
```

10.177.1.383 CKM_ECDH1_DERIVE

```
#define CKM_ECDH1_DERIVE 0x00001050UL
```

10.177.1.384 CKM_ECDH_AES_KEY_WRAP

```
#define CKM_ECDH_AES_KEY_WRAP 0x00001053UL
```

10.177.1.385 CKM_ECDSA

```
#define CKM_ECDSA 0x00001041UL
```

10.177.1.386 CKM_ECDSA_KEY_PAIR_GEN

```
#define CKM_ECDSA_KEY_PAIR_GEN 0x00001040UL /* Deprecated */
```

10.177.1.387 CKM_ECDSA_SHA1

```
#define CKM_ECDSA_SHA1 0x00001042UL
```

10.177.1.388 CKM_ECDSA_SHA224

```
#define CKM_ECDSA_SHA224 0x00001043UL
```

10.177.1.389 CKM_ECDSA_SHA256

```
#define CKM_ECDSA_SHA256 0x00001044UL
```

10.177.1.390 CKM_ECDSA_SHA384

```
#define CKM_ECDSA_SHA384 0x00001045UL
```

10.177.1.391 CKM_ECDSA_SHA512

```
#define CKM_ECDSA_SHA512 0x00001046UL
```

10.177.1.392 CKM_ECMQV_DERIVE

```
#define CKM_ECMQV_DERIVE 0x00001052UL
```

10.177.1.393 CKM_EXTRACT_KEY_FROM_KEY

```
#define CKM_EXTRACT_KEY_FROM_KEY 0x00000365UL
```

10.177.1.394 CKM_FASTHASH

```
#define CKM_FASTHASH 0x00001070UL
```

10.177.1.395 CKM_FORTEZZA_TIMESTAMP

```
#define CKM_FORTEZZA_TIMESTAMP 0x00001020UL
```

10.177.1.396 CKM_GENERIC_SECRET_KEY_GEN

```
#define CKM_GENERIC_SECRET_KEY_GEN 0x00000350UL
```

10.177.1.397 CKM_GOST28147

```
#define CKM_GOST28147 0x00001222UL
```

10.177.1.398 CKM_GOST28147_ECB

```
#define CKM_GOST28147_ECB 0x00001221UL
```

10.177.1.399 CKM_GOST28147_KEY_GEN

```
#define CKM_GOST28147_KEY_GEN 0x00001220UL
```

10.177.1.400 CKM_GOST28147_KEY_WRAP

```
#define CKM_GOST28147_KEY_WRAP 0x00001224UL
```

10.177.1.401 CKM_GOST28147_MAC

```
#define CKM_GOST28147_MAC 0x00001223UL
```

10.177.1.402 CKM_GOSTR3410

```
#define CKM_GOSTR3410 0x00001201UL
```

10.177.1.403 CKM_GOSTR3410_DERIVE

```
#define CKM_GOSTR3410_DERIVE 0x00001204UL
```

10.177.1.404 CKM_GOSTR3410_KEY_PAIR_GEN

```
#define CKM_GOSTR3410_KEY_PAIR_GEN 0x00001200UL
```

10.177.1.405 CKM_GOSTR3410_KEY_WRAP

```
#define CKM_GOSTR3410_KEY_WRAP 0x00001203UL
```

10.177.1.406 CKM_GOSTR3410_WITH_GOSTR3411

```
#define CKM_GOSTR3410_WITH_GOSTR3411 0x00001202UL
```

10.177.1.407 CKM_GOSTR3411

```
#define CKM_GOSTR3411 0x00001210UL
```

10.177.1.408 CKM_GOSTR3411_HMAC

```
#define CKM_GOSTR3411_HMAC 0x00001211UL
```

10.177.1.409 CKM_HOTP

```
#define CKM_HOTP 0x00000291UL
```

10.177.1.410 CKM_HOTP_KEY_GEN

```
#define CKM_HOTP_KEY_GEN 0x00000290UL
```

10.177.1.411 CKM_IDEA_CBC

```
#define CKM_IDEA_CBC 0x00000342UL
```

10.177.1.412 CKM_IDEA_CBC_PAD

```
#define CKM_IDEA_CBC_PAD 0x00000345UL
```

10.177.1.413 CKM_IDEA_ECB

```
#define CKM_IDEA_ECB 0x00000341UL
```

10.177.1.414 CKM_IDEA_KEY_GEN

```
#define CKM_IDEA_KEY_GEN 0x00000340UL
```

10.177.1.415 CKM_IDEA_MAC

```
#define CKM_IDEA_MAC 0x00000343UL
```

10.177.1.416 CKM_IDEA_MAC_GENERAL

```
#define CKM_IDEA_MAC_GENERAL 0x00000344UL
```

10.177.1.417 CKM_JUNIPER_CBC128

```
#define CKM_JUNIPER_CBC128 0x00001062UL
```

10.177.1.418 CKM_JUNIPER_COUNTER

```
#define CKM_JUNIPER_COUNTER 0x00001063UL
```

10.177.1.419 CKM_JUNIPER_ECB128

```
#define CKM_JUNIPER_ECB128 0x00001061UL
```

10.177.1.420 CKM_JUNIPER_KEY_GEN

```
#define CKM_JUNIPER_KEY_GEN 0x00001060UL
```

10.177.1.421 CKM_JUNIPER_SHUFFLE

```
#define CKM_JUNIPER_SHUFFLE 0x00001064UL
```

10.177.1.422 CKM_JUNIPER_WRAP

```
#define CKM_JUNIPER_WRAP 0x00001065UL
```

10.177.1.423 CKM_KEA_DERIVE

```
#define CKM_KEA_DERIVE 0x00001012UL
```

10.177.1.424 CKM_KEA_KEY_DERIVE

```
#define CKM_KEA_KEY_DERIVE 0x00001011UL
```

10.177.1.425 CKM_KEA_KEY_PAIR_GEN

```
#define CKM_KEA_KEY_PAIR_GEN 0x00001010UL
```

10.177.1.426 CKM_KEY_WRAP_LYNKS

```
#define CKM_KEY_WRAP_LYNKS 0x00000400UL
```

10.177.1.427 CKM_KEY_WRAP_SET_OAEP

```
#define CKM_KEY_WRAP_SET_OAEP 0x00000401UL
```

10.177.1.428 CKM_KIP_DERIVE

```
#define CKM_KIP_DERIVE 0x00000510UL
```

10.177.1.429 CKM_KIP_MAC

```
#define CKM_KIP_MAC 0x00000512UL
```

10.177.1.430 CKM_KIP_WRAP

```
#define CKM_KIP_WRAP 0x00000511UL
```

10.177.1.431 CKM_MD2

```
#define CKM_MD2 0x00000200UL
```

10.177.1.432 CKM_MD2_HMAC

```
#define CKM_MD2_HMAC 0x00000201UL
```

10.177.1.433 CKM_MD2_HMAC_GENERAL

```
#define CKM_MD2_HMAC_GENERAL 0x00000202UL
```

10.177.1.434 CKM_MD2_KEY_DERIVATION

```
#define CKM_MD2_KEY_DERIVATION 0x00000391UL
```

10.177.1.435 CKM_MD2_RSA_PKCS

```
#define CKM_MD2_RSA_PKCS 0x00000004UL
```

10.177.1.436 CKM_MD5

```
#define CKM_MD5 0x00000210UL
```

10.177.1.437 CKM_MD5_HMAC

```
#define CKM_MD5_HMAC 0x00000211UL
```

10.177.1.438 CKM_MD5_HMAC_GENERAL

```
#define CKM_MD5_HMAC_GENERAL 0x00000212UL
```

10.177.1.439 CKM_MD5_KEY_DERIVATION

```
#define CKM_MD5_KEY_DERIVATION 0x00000390UL
```

10.177.1.440 CKM_MD5_RSA_PKCS

```
#define CKM_MD5_RSA_PKCS 0x00000005UL
```

10.177.1.441 CKM_PBA_SHA1_WITH_SHA1_HMAC

```
#define CKM_PBA_SHA1_WITH_SHA1_HMAC 0x000003C0UL
```

10.177.1.442 CKM_PBE_MD2_DES_CBC

```
#define CKM_PBE_MD2_DES_CBC 0x000003A0UL
```

10.177.1.443 CKM_PBE_MD5_CAST128_CBC

```
#define CKM_PBE_MD5_CAST128_CBC 0x000003A4UL
```

10.177.1.444 CKM_PBE_MD5_CAST3_CBC

```
#define CKM_PBE_MD5_CAST3_CBC 0x000003A3UL
```

10.177.1.445 CKM_PBE_MD5_CAST5_CBC

```
#define CKM_PBE_MD5_CAST5_CBC 0x000003A4UL /* Deprecated */
```

10.177.1.446 CKM_PBE_MD5_CAST_CBC

```
#define CKM_PBE_MD5_CAST_CBC 0x000003A2UL
```

10.177.1.447 CKM_PBE_MD5_DES_CBC

```
#define CKM_PBE_MD5_DES_CBC 0x000003A1UL
```

10.177.1.448 CKM_PBE_SHA1_CAST128_CBC

```
#define CKM_PBE_SHA1_CAST128_CBC 0x000003A5UL
```

10.177.1.449 CKM_PBE_SHA1_CAST5_CBC

```
#define CKM_PBE_SHA1_CAST5_CBC 0x000003A5UL /* Deprecated */
```

10.177.1.450 CKM_PBE_SHA1_DES2_EDE_CBC

```
#define CKM_PBE_SHA1_DES2_EDE_CBC 0x000003A9UL
```

10.177.1.451 CKM_PBE_SHA1_DES3_EDE_CBC

```
#define CKM_PBE_SHA1_DES3_EDE_CBC 0x000003A8UL
```

10.177.1.452 CKM_PBE_SHA1_RC2_128_CBC

```
#define CKM_PBE_SHA1_RC2_128_CBC 0x000003AAUL
```

10.177.1.453 CKM_PBE_SHA1_RC2_40_CBC

```
#define CKM_PBE_SHA1_RC2_40_CBC 0x000003ABUL
```

10.177.1.454 CKM_PBE_SHA1_RC4_128

```
#define CKM_PBE_SHA1_RC4_128 0x000003A6UL
```

10.177.1.455 CKM_PBE_SHA1_RC4_40

```
#define CKM_PBE_SHA1_RC4_40 0x000003A7UL
```

10.177.1.456 CKM_PKCS5_PBKD2

```
#define CKM_PKCS5_PBKD2 0x000003B0UL
```

10.177.1.457 CKM_RC2_CBC

```
#define CKM_RC2_CBC 0x00000102UL
```

10.177.1.458 CKM_RC2_CBC_PAD

```
#define CKM_RC2_CBC_PAD 0x00000105UL
```

10.177.1.459 CKM_RC2_ECB

```
#define CKM_RC2_ECB 0x00000101UL
```

10.177.1.460 CKM_RC2_KEY_GEN

```
#define CKM_RC2_KEY_GEN 0x00000100UL
```

10.177.1.461 CKM_RC2_MAC

```
#define CKM_RC2_MAC 0x00000103UL
```

10.177.1.462 CKM_RC2_MAC_GENERAL

```
#define CKM_RC2_MAC_GENERAL 0x00000104UL
```

10.177.1.463 CKM_RC4

```
#define CKM_RC4 0x00000111UL
```

10.177.1.464 CKM_RC4_KEY_GEN

```
#define CKM_RC4_KEY_GEN 0x00000110UL
```

10.177.1.465 CKM_RC5_CBC

```
#define CKM_RC5_CBC 0x00000332UL
```

10.177.1.466 CKM_RC5_CBC_PAD

```
#define CKM_RC5_CBC_PAD 0x00000335UL
```

10.177.1.467 CKM_RC5_ECB

```
#define CKM_RC5_ECB 0x00000331UL
```

10.177.1.468 CKM_RC5_KEY_GEN

```
#define CKM_RC5_KEY_GEN 0x00000330UL
```

10.177.1.469 CKM_RC5_MAC

```
#define CKM_RC5_MAC 0x00000333UL
```

10.177.1.470 CKM_RC5_MAC_GENERAL

```
#define CKM_RC5_MAC_GENERAL 0x00000334UL
```

10.177.1.471 CKM_RIPEMD128

```
#define CKM_RIPEMD128 0x00000230UL
```

10.177.1.472 CKM_RIPEMD128_HMAC

```
#define CKM_RIPEMD128_HMAC 0x00000231UL
```

10.177.1.473 CKM_RIPEMD128_HMAC_GENERAL

```
#define CKM_RIPEMD128_HMAC_GENERAL 0x00000232UL
```

10.177.1.474 CKM_RIPEMD128_RSA_PKCS

```
#define CKM_RIPEMD128_RSA_PKCS 0x00000007UL
```

10.177.1.475 CKM_RIPEMD160

```
#define CKM_RIPEMD160 0x00000240UL
```

10.177.1.476 CKM_RIPEMD160_HMAC

```
#define CKM_RIPEMD160_HMAC 0x00000241UL
```

10.177.1.477 CKM_RIPEMD160_HMAC_GENERAL

```
#define CKM_RIPEMD160_HMAC_GENERAL 0x00000242UL
```

10.177.1.478 CKM_RIPEMD160_RSA_PKCS

```
#define CKM_RIPEMD160_RSA_PKCS 0x00000008UL
```

10.177.1.479 CKM_RSA_9796

```
#define CKM_RSA_9796 0x00000002UL
```

10.177.1.480 CKM_RSA_AES_KEY_WRAP

```
#define CKM_RSA_AES_KEY_WRAP 0x00001054UL
```

10.177.1.481 CKM_RSA_PKCS

```
#define CKM_RSA_PKCS 0x00000001UL
```

10.177.1.482 CKM_RSA_PKCS_KEY_PAIR_GEN

```
#define CKM_RSA_PKCS_KEY_PAIR_GEN 0x00000000UL
```

10.177.1.483 CKM_RSA_PKCS_OAEP

```
#define CKM_RSA_PKCS_OAEP 0x00000009UL
```

10.177.1.484 CKM_RSA_PKCS_OAEP_TPM_1_1

```
#define CKM_RSA_PKCS_OAEP_TPM_1_1 0x00004002UL
```

10.177.1.485 CKM_RSA_PKCS_PSS

```
#define CKM_RSA_PKCS_PSS 0x0000000DUL
```

10.177.1.486 CKM_RSA_PKCS_TPM_1_1

```
#define CKM_RSA_PKCS_TPM_1_1 0x00004001UL
```

10.177.1.487 CKM_RSA_X9_31

```
#define CKM_RSA_X9_31 0x0000000BUL
```

10.177.1.488 CKM_RSA_X9_31_KEY_PAIR_GEN

```
#define CKM_RSA_X9_31_KEY_PAIR_GEN 0x0000000AUL
```

10.177.1.489 CKM_RSA_X_509

```
#define CKM_RSA_X_509 0x00000003UL
```

10.177.1.490 CKM_SECURID

```
#define CKM_SECURID 0x00000282UL
```

10.177.1.491 CKM_SECURID_KEY_GEN

```
#define CKM_SECURID_KEY_GEN 0x00000280UL
```

10.177.1.492 CKM_SEED_CBC

```
#define CKM_SEED_CBC 0x00000652UL
```

10.177.1.493 CKM_SEED_CBC_ENCRYPT_DATA

```
#define CKM_SEED_CBC_ENCRYPT_DATA 0x00000657UL
```

10.177.1.494 CKM_SEED_CBC_PAD

```
#define CKM_SEED_CBC_PAD 0x00000655UL
```

10.177.1.495 CKM_SEED_ECB

```
#define CKM_SEED_ECB 0x00000651UL
```

10.177.1.496 CKM_SEED_ECB_ENCRYPT_DATA

```
#define CKM_SEED_ECB_ENCRYPT_DATA 0x00000656UL
```

10.177.1.497 CKM_SEED_KEY_GEN

```
#define CKM_SEED_KEY_GEN 0x00000650UL
```

10.177.1.498 CKM_SEED_MAC

```
#define CKM_SEED_MAC 0x00000653UL
```

10.177.1.499 CKM_SEED_MAC_GENERAL

```
#define CKM_SEED_MAC_GENERAL 0x00000654UL
```

10.177.1.500 CKM_SHA1_KEY_DERIVATION

```
#define CKM_SHA1_KEY_DERIVATION 0x00000392UL
```

10.177.1.501 CKM_SHA1_RSA_PKCS

```
#define CKM_SHA1_RSA_PKCS 0x00000006UL
```

10.177.1.502 CKM_SHA1_RSA_PKCS_PSS

```
#define CKM_SHA1_RSA_PKCS_PSS 0x0000000EUL
```

10.177.1.503 CKM_SHA1_RSA_X9_31

```
#define CKM_SHA1_RSA_X9_31 0x0000000CUL
```

10.177.1.504 CKM_SHA224

```
#define CKM_SHA224 0x00000255UL
```

10.177.1.505 CKM_SHA224_HMAC

```
#define CKM_SHA224_HMAC 0x00000256UL
```

10.177.1.506 CKM_SHA224_HMAC_GENERAL

```
#define CKM_SHA224_HMAC_GENERAL 0x00000257UL
```

10.177.1.507 CKM_SHA224_KEY_DERIVATION

```
#define CKM_SHA224_KEY_DERIVATION 0x00000396UL
```

10.177.1.508 CKM_SHA224_RSA_PKCS

```
#define CKM_SHA224_RSA_PKCS 0x00000046UL
```

10.177.1.509 CKM_SHA224_RSA_PKCS_PSS

```
#define CKM_SHA224_RSA_PKCS_PSS 0x00000047UL
```

10.177.1.510 CKM_SHA256

```
#define CKM_SHA256 0x00000250UL
```

10.177.1.511 CKM_SHA256_HMAC

```
#define CKM_SHA256_HMAC 0x00000251UL
```

10.177.1.512 CKM_SHA256_HMAC_GENERAL

```
#define CKM_SHA256_HMAC_GENERAL 0x00000252UL
```

10.177.1.513 CKM_SHA256_KEY_DERIVATION

```
#define CKM_SHA256_KEY_DERIVATION 0x00000393UL
```

10.177.1.514 CKM_SHA256_RSA_PKCS

```
#define CKM_SHA256_RSA_PKCS 0x00000040UL
```

10.177.1.515 CKM_SHA256_RSA_PKCS_PSS

```
#define CKM_SHA256_RSA_PKCS_PSS 0x00000043UL
```

10.177.1.516 CKM_SHA384

```
#define CKM_SHA384 0x00000260UL
```

10.177.1.517 CKM_SHA384_HMAC

```
#define CKM_SHA384_HMAC 0x00000261UL
```

10.177.1.518 CKM_SHA384_HMAC_GENERAL

```
#define CKM_SHA384_HMAC_GENERAL 0x00000262UL
```

10.177.1.519 CKM_SHA384_KEY_DERIVATION

```
#define CKM_SHA384_KEY_DERIVATION 0x00000394UL
```

10.177.1.520 CKM_SHA384_RSA_PKCS

```
#define CKM_SHA384_RSA_PKCS 0x00000041UL
```

10.177.1.521 CKM_SHA384_RSA_PKCS_PSS

```
#define CKM_SHA384_RSA_PKCS_PSS 0x00000044UL
```

10.177.1.522 CKM_SHA512

```
#define CKM_SHA512 0x00000270UL
```

10.177.1.523 CKM_SHA512_224

```
#define CKM_SHA512_224 0x00000048UL
```

10.177.1.524 CKM_SHA512_224_HMAC

```
#define CKM_SHA512_224_HMAC 0x00000049UL
```

10.177.1.525 CKM_SHA512_224_HMAC_GENERAL

```
#define CKM_SHA512_224_HMAC_GENERAL 0x0000004AUL
```

10.177.1.526 CKM_SHA512_224_KEY_DERIVATION

```
#define CKM_SHA512_224_KEY_DERIVATION 0x0000004BUL
```

10.177.1.527 CKM_SHA512_256

```
#define CKM_SHA512_256 0x0000004CUL
```

10.177.1.528 CKM_SHA512_256_HMAC

```
#define CKM_SHA512_256_HMAC 0x0000004DUL
```

10.177.1.529 CKM_SHA512_256_HMAC_GENERAL

```
#define CKM_SHA512_256_HMAC_GENERAL 0x0000004EUL
```

10.177.1.530 CKM_SHA512_256_KEY_DERIVATION

```
#define CKM_SHA512_256_KEY_DERIVATION 0x0000004FUL
```

10.177.1.531 CKM_SHA512_HMAC

```
#define CKM_SHA512_HMAC 0x00000271UL
```

10.177.1.532 CKM_SHA512_HMAC_GENERAL

```
#define CKM_SHA512_HMAC_GENERAL 0x00000272UL
```

10.177.1.533 CKM_SHA512_KEY_DERIVATION

```
#define CKM_SHA512_KEY_DERIVATION 0x00000395UL
```

10.177.1.534 CKM_SHA512_RSA_PKCS

```
#define CKM_SHA512_RSA_PKCS 0x00000042UL
```

10.177.1.535 CKM_SHA512_RSA_PKCS_PSS

```
#define CKM_SHA512_RSA_PKCS_PSS 0x00000045UL
```

10.177.1.536 CKM_SHA512_T

```
#define CKM_SHA512_T 0x00000050UL
```

10.177.1.537 CKM_SHA512_T_HMAC

```
#define CKM_SHA512_T_HMAC 0x00000051UL
```

10.177.1.538 CKM_SHA512_T_HMAC_GENERAL

```
#define CKM_SHA512_T_HMAC_GENERAL 0x00000052UL
```

10.177.1.539 CKM_SHA512_T_KEY_DERIVATION

```
#define CKM_SHA512_T_KEY_DERIVATION 0x00000053UL
```

10.177.1.540 CKM_SHA_1

```
#define CKM_SHA_1 0x00000220UL
```

10.177.1.541 CKM_SHA_1_HMAC

```
#define CKM_SHA_1_HMAC 0x00000221UL
```

10.177.1.542 CKM_SHA_1_HMAC_GENERAL

```
#define CKM_SHA_1_HMAC_GENERAL 0x00000222UL
```

10.177.1.543 CKM_SKIPJACK_CBC64

```
#define CKM_SKIPJACK_CBC64 0x00001002UL
```

10.177.1.544 CKM_SKIPJACK_CFB16

```
#define CKM_SKIPJACK_CFB16 0x00001006UL
```

10.177.1.545 CKM_SKIPJACK_CFB32

```
#define CKM_SKIPJACK_CFB32 0x00001005UL
```

10.177.1.546 CKM_SKIPJACK_CFB64

```
#define CKM_SKIPJACK_CFB64 0x00001004UL
```

10.177.1.547 CKM_SKIPJACK_CFB8

```
#define CKM_SKIPJACK_CFB8 0x00001007UL
```

10.177.1.548 CKM_SKIPJACK_ECB64

```
#define CKM_SKIPJACK_ECB64 0x00001001UL
```

10.177.1.549 CKM_SKIPJACK_KEY_GEN

```
#define CKM_SKIPJACK_KEY_GEN 0x00001000UL
```

10.177.1.550 CKM_SKIPJACK_OFB64

```
#define CKM_SKIPJACK_OFB64 0x00001003UL
```

10.177.1.551 CKM_SKIPJACK_PRIVATE_WRAP

```
#define CKM_SKIPJACK_PRIVATE_WRAP 0x00001009UL
```

10.177.1.552 CKM_SKIPJACK_RELAYX

```
#define CKM_SKIPJACK_RELAYX 0x0000100aUL
```

10.177.1.553 CKM_SKIPJACK_WRAP

```
#define CKM_SKIPJACK_WRAP 0x00001008UL
```

10.177.1.554 CKM_SSL3_KEY_AND_MAC_DERIVE

```
#define CKM_SSL3_KEY_AND_MAC_DERIVE 0x00000372UL
```

10.177.1.555 CKM_SSL3_MASTER_KEY_DERIVE

```
#define CKM_SSL3_MASTER_KEY_DERIVE 0x00000371UL
```

10.177.1.556 CKM_SSL3_MASTER_KEY_DERIVE_DH

```
#define CKM_SSL3_MASTER_KEY_DERIVE_DH 0x00000373UL
```

10.177.1.557 CKM_SSL3_MD5_MAC

```
#define CKM_SSL3_MD5_MAC 0x00000380UL
```

10.177.1.558 CKM_SSL3_PRE_MASTER_KEY_GEN

```
#define CKM_SSL3_PRE_MASTER_KEY_GEN 0x00000370UL
```

10.177.1.559 CKM_SSL3_SHA1_MAC

```
#define CKM_SSL3_SHA1_MAC 0x00000381UL
```

10.177.1.560 CKM_TLS10_MAC_CLIENT

```
#define CKM_TLS10_MAC_CLIENT 0x000003D7UL
```

10.177.1.561 CKM_TLS10_MAC_SERVER

```
#define CKM_TLS10_MAC_SERVER 0x000003D6UL
```

10.177.1.562 CKM_TLS12_KDF

```
#define CKM_TLS12_KDF 0x000003D9UL
```

10.177.1.563 CKM_TLS12_KEY_AND_MAC_DERIVE

```
#define CKM_TLS12_KEY_AND_MAC_DERIVE 0x000003E1UL
```

10.177.1.564 CKM_TLS12_KEY_SAFE_DERIVE

```
#define CKM_TLS12_KEY_SAFE_DERIVE 0x000003E3UL
```

10.177.1.565 CKM_TLS12_MAC

```
#define CKM_TLS12_MAC 0x000003D8UL
```

10.177.1.566 CKM_TLS12_MASTER_KEY_DERIVE

```
#define CKM_TLS12_MASTER_KEY_DERIVE 0x000003E0UL
```

10.177.1.567 CKM_TLS12_MASTER_KEY_DERIVE_DH

```
#define CKM_TLS12_MASTER_KEY_DERIVE_DH 0x000003E2UL
```

10.177.1.568 CKM_TLS_KDF

```
#define CKM_TLS_KDF 0x000003E5UL
```

10.177.1.569 CKM_TLS_KEY_AND_MAC_DERIVE

```
#define CKM_TLS_KEY_AND_MAC_DERIVE 0x00000376UL
```

10.177.1.570 CKM_TLS_MAC

```
#define CKM_TLS_MAC 0x000003E4UL
```

10.177.1.571 CKM_TLS_MASTER_KEY_DERIVE

```
#define CKM_TLS_MASTER_KEY_DERIVE 0x00000375UL
```

10.177.1.572 CKM_TLS_MASTER_KEY_DERIVE_DH

```
#define CKM_TLS_MASTER_KEY_DERIVE_DH 0x00000377UL
```

10.177.1.573 CKM_TLS_PRE_MASTER_KEY_GEN

```
#define CKM_TLS_PRE_MASTER_KEY_GEN 0x00000374UL
```

10.177.1.574 CKM_TLS_PRF

```
#define CKM_TLS_PRF 0x00000378UL
```

10.177.1.575 CKM_TWOFISH_CBC

```
#define CKM_TWOFISH_CBC 0x00001093UL
```

10.177.1.576 CKM_TWOFISH_CBC_PAD

```
#define CKM_TWOFISH_CBC_PAD 0x00001095UL
```

10.177.1.577 CKM_TWOFISH_KEY_GEN

```
#define CKM_TWOFISH_KEY_GEN 0x00001092UL
```

10.177.1.578 CKM_VENDOR_DEFINED

```
#define CKM_VENDOR_DEFINED 0x80000000UL
```

10.177.1.579 CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE

```
#define CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE 0x000003D5UL
```

10.177.1.580 CKM_WTLS_MASTER_KEY_DERIVE

```
#define CKM_WTLS_MASTER_KEY_DERIVE 0x000003D1UL
```

10.177.1.581 CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC

```
#define CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC 0x000003D2UL
```

10.177.1.582 CKM_WTLS_PRE_MASTER_KEY_GEN

```
#define CKM_WTLS_PRE_MASTER_KEY_GEN 0x000003D0UL
```

10.177.1.583 CKM_WTLS_PRF

```
#define CKM_WTLS_PRF 0x000003D3UL
```

10.177.1.584 CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE

```
#define CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE 0x000003D4UL
```

10.177.1.585 CKM_X9_42_DH_DERIVE

```
#define CKM_X9_42_DH_DERIVE 0x00000031UL
```

10.177.1.586 CKM_X9_42_DH_HYBRID_DERIVE

```
#define CKM_X9_42_DH_HYBRID_DERIVE 0x00000032UL
```

10.177.1.587 CKM_X9_42_DH_KEY_PAIR_GEN

```
#define CKM_X9_42_DH_KEY_PAIR_GEN 0x00000030UL
```

10.177.1.588 CKM_X9_42_DH_PARAMETER_GEN

```
#define CKM_X9_42_DH_PARAMETER_GEN 0x00002002UL
```

10.177.1.589 CKM_X9_42_MQV_DERIVE

```
#define CKM_X9_42_MQV_DERIVE 0x00000033UL
```

10.177.1.590 CKM_XOR_BASE_AND_DATA

```
#define CKM_XOR_BASE_AND_DATA 0x00000364UL
```

10.177.1.591 CKN_OTP_CHANGED

```
#define CKN_OTP_CHANGED 1UL
```

10.177.1.592 CKN_SURRENDER

```
#define CKN_SURRENDER 0UL
```

10.177.1.593 CKO_CERTIFICATE

```
#define CKO_CERTIFICATE 0x00000001UL
```

10.177.1.594 CKO_DATA

```
#define CKO_DATA 0x00000000UL
```

10.177.1.595 CKO_DOMAIN_PARAMETERS

```
#define CKO_DOMAIN_PARAMETERS 0x00000006UL
```

10.177.1.596 CKO_HW_FEATURE

```
#define CKO_HW_FEATURE 0x00000005UL
```

10.177.1.597 CKO_MECHANISM

```
#define CKO_MECHANISM 0x00000007UL
```

10.177.1.598 CKO_OTP_KEY

```
#define CKO_OTP_KEY 0x00000008UL
```

10.177.1.599 CKO_PRIVATE_KEY

```
#define CKO_PRIVATE_KEY 0x00000003UL
```

10.177.1.600 CKO_PUBLIC_KEY

```
#define CKO_PUBLIC_KEY 0x00000002UL
```

10.177.1.601 CKO_SECRET_KEY

```
#define CKO_SECRET_KEY 0x00000004UL
```

10.177.1.602 CKO_VENDOR_DEFINED

```
#define CKO_VENDOR_DEFINED 0x80000000UL
```

10.177.1.603 CKP_PKCS5_PBKD2_HMAC_GOSTR3411

```
#define CKP_PKCS5_PBKD2_HMAC_GOSTR3411 0x00000002UL
```

10.177.1.604 CKP_PKCS5_PBKD2_HMAC_SHA1

```
#define CKP_PKCS5_PBKD2_HMAC_SHA1 0x00000001UL
```

10.177.1.605 CKP_PKCS5_PBKD2_HMAC_SHA224

```
#define CKP_PKCS5_PBKD2_HMAC_SHA224 0x00000003UL
```

10.177.1.606 CKP_PKCS5_PBKD2_HMAC_SHA256

```
#define CKP_PKCS5_PBKD2_HMAC_SHA256 0x00000004UL
```

10.177.1.607 CKP_PKCS5_PBKD2_HMAC_SHA384

```
#define CKP_PKCS5_PBKD2_HMAC_SHA384 0x00000005UL
```

10.177.1.608 CKP_PKCS5_PBKD2_HMAC_SHA512

```
#define CKP_PKCS5_PBKD2_HMAC_SHA512 0x00000006UL
```

10.177.1.609 CKP_PKCS5_PBKD2_HMAC_SHA512_224

```
#define CKP_PKCS5_PBKD2_HMAC_SHA512_224 0x00000007UL
```

10.177.1.610 CKP_PKCS5_PBKD2_HMAC_SHA512_256

```
#define CKP_PKCS5_PBKD2_HMAC_SHA512_256 0x00000008UL
```

10.177.1.611 CKR_ACTION_PROHIBITED

```
#define CKR_ACTION_PROHIBITED 0x0000001BUL
```

10.177.1.612 CKR_ARGUMENTS_BAD

```
#define CKR_ARGUMENTS_BAD 0x00000007UL
```

10.177.1.613 CKR_ATTRIBUTE_READ_ONLY

```
#define CKR_ATTRIBUTE_READ_ONLY 0x00000010UL
```

10.177.1.614 CKR_ATTRIBUTE_SENSITIVE

```
#define CKR_ATTRIBUTE_SENSITIVE 0x00000011UL
```

10.177.1.615 CKR_ATTRIBUTE_TYPE_INVALID

```
#define CKR_ATTRIBUTE_TYPE_INVALID 0x00000012UL
```

10.177.1.616 CKR_ATTRIBUTE_VALUE_INVALID

```
#define CKR_ATTRIBUTE_VALUE_INVALID 0x00000013UL
```

10.177.1.617 CKR_BUFFER_TOO_SMALL

```
#define CKR_BUFFER_TOO_SMALL 0x00000150UL
```

10.177.1.618 CKR_CANCEL

```
#define CKR_CANCEL 0x00000001UL
```

10.177.1.619 CKR_CANT_LOCK

```
#define CKR_CANT_LOCK 0x0000000AUL
```

10.177.1.620 CKR_CRYPTOKI_ALREADY_INITIALIZED

```
#define CKR_CRYPTOKI_ALREADY_INITIALIZED 0x00000191UL
```

10.177.1.621 CKR_CRYPTOKI_NOT_INITIALIZED

```
#define CKR_CRYPTOKI_NOT_INITIALIZED 0x00000190UL
```

10.177.1.622 CKR_CURVE_NOT_SUPPORTED

```
#define CKR_CURVE_NOT_SUPPORTED 0x00000140UL
```

10.177.1.623 CKR_DATA_INVALID

```
#define CKR_DATA_INVALID 0x00000020UL
```

10.177.1.624 CKR_DATA_LEN_RANGE

```
#define CKR_DATA_LEN_RANGE 0x00000021UL
```

10.177.1.625 CKR_DEVICE_ERROR

```
#define CKR_DEVICE_ERROR 0x00000030UL
```

10.177.1.626 CKR_DEVICE_MEMORY

```
#define CKR_DEVICE_MEMORY 0x00000031UL
```

10.177.1.627 CKR_DEVICE_REMOVED

```
#define CKR_DEVICE_REMOVED 0x00000032UL
```

10.177.1.628 CKR_DOMAIN_PARAMS_INVALID

```
#define CKR_DOMAIN_PARAMS_INVALID 0x00000130UL
```

10.177.1.629 CKR_ENCRYPTED_DATA_INVALID

```
#define CKR_ENCRYPTED_DATA_INVALID 0x00000040UL
```

10.177.1.630 CKR_ENCRYPTED_DATA_LEN_RANGE

```
#define CKR_ENCRYPTED_DATA_LEN_RANGE 0x00000041UL
```

10.177.1.631 CKR_EXCEEDED_MAX_ITERATIONS

```
#define CKR_EXCEEDED_MAX_ITERATIONS 0x000001B5UL
```

10.177.1.632 CKR_FIPS_SELF_TEST_FAILED

```
#define CKR_FIPS_SELF_TEST_FAILED 0x000001B6UL
```

10.177.1.633 CKR_FUNCTION_CANCELED

```
#define CKR_FUNCTION_CANCELED 0x00000050UL
```

10.177.1.634 CKR_FUNCTION_FAILED

```
#define CKR_FUNCTION_FAILED 0x00000006UL
```

10.177.1.635 CKR_FUNCTION_NOT_PARALLEL

```
#define CKR_FUNCTION_NOT_PARALLEL 0x00000051UL
```

10.177.1.636 CKR_FUNCTION_NOT_SUPPORTED

```
#define CKR_FUNCTION_NOT_SUPPORTED 0x00000054UL
```

10.177.1.637 CKR_FUNCTION_REJECTED

```
#define CKR_FUNCTION_REJECTED 0x00000200UL
```

10.177.1.638 CKR_GENERAL_ERROR

```
#define CKR_GENERAL_ERROR 0x00000005UL
```

10.177.1.639 CKR_HOST_MEMORY

```
#define CKR_HOST_MEMORY 0x00000002UL
```

10.177.1.640 CKR_INFORMATION_SENSITIVE

```
#define CKR_INFORMATION_SENSITIVE 0x00000170UL
```

10.177.1.641 CKR_KEY_CHANGED

```
#define CKR_KEY_CHANGED 0x00000065UL
```

10.177.1.642 CKR_KEY_FUNCTION_NOT_PERMITTED

```
#define CKR_KEY_FUNCTION_NOT_PERMITTED 0x00000068UL
```

10.177.1.643 CKR_KEY_HANDLE_INVALID

```
#define CKR_KEY_HANDLE_INVALID 0x00000060UL
```

10.177.1.644 CKR_KEY_INDIGESTIBLE

```
#define CKR_KEY_INDIGESTIBLE 0x00000067UL
```

10.177.1.645 CKR_KEY_NEEDED

```
#define CKR_KEY_NEEDED 0x00000066UL
```

10.177.1.646 CKR_KEY_NOT_NEEDED

```
#define CKR_KEY_NOT_NEEDED 0x00000064UL
```

10.177.1.647 CKR_KEY_NOT_WRAPPABLE

```
#define CKR_KEY_NOT_WRAPPABLE 0x00000069UL
```

10.177.1.648 CKR_KEY_SIZE_RANGE

```
#define CKR_KEY_SIZE_RANGE 0x00000062UL
```

10.177.1.649 CKR_KEY_TYPE_INCONSISTENT

```
#define CKR_KEY_TYPE_INCONSISTENT 0x00000063UL
```

10.177.1.650 CKR_KEY_UNEXTRACTABLE

```
#define CKR_KEY_UNEXTRACTABLE 0x0000006AUL
```

10.177.1.651 CKR_LIBRARY_LOAD_FAILED

```
#define CKR_LIBRARY_LOAD_FAILED 0x000001B7UL
```

10.177.1.652 CKR_MECHANISM_INVALID

```
#define CKR_MECHANISM_INVALID 0x00000070UL
```

10.177.1.653 CKR_MECHANISM_PARAM_INVALID

```
#define CKR_MECHANISM_PARAM_INVALID 0x00000071UL
```

10.177.1.654 CKR_MUTEX_BAD

```
#define CKR_MUTEX_BAD 0x000001A0UL
```

10.177.1.655 CKR_MUTEX_NOT_LOCKED

```
#define CKR_MUTEX_NOT_LOCKED 0x000001A1UL
```

10.177.1.656 CKR_NEED_TO_CREATE_THREADS

```
#define CKR_NEED_TO_CREATE_THREADS 0x00000009UL
```

10.177.1.657 CKR_NEW_PIN_MODE

```
#define CKR_NEW_PIN_MODE 0x000001B0UL
```

10.177.1.658 CKR_NEXT_OTP

```
#define CKR_NEXT_OTP 0x000001B1UL
```

10.177.1.659 CKR_NO_EVENT

```
#define CKR_NO_EVENT 0x00000008UL
```

10.177.1.660 CKR_OBJECT_HANDLE_INVALID

```
#define CKR_OBJECT_HANDLE_INVALID 0x00000082UL
```

10.177.1.661 CKR_OK

```
#define CKR_OK 0x00000000UL
```

10.177.1.662 CKR_OPERATION_ACTIVE

```
#define CKR_OPERATION_ACTIVE 0x00000090UL
```

10.177.1.663 CKR_OPERATION_NOT_INITIALIZED

```
#define CKR_OPERATION_NOT_INITIALIZED 0x00000091UL
```

10.177.1.664 CKR_PIN_EXPIRED

```
#define CKR_PIN_EXPIRED 0x000000A3UL
```

10.177.1.665 CKR_PIN_INCORRECT

```
#define CKR_PIN_INCORRECT 0x000000A0UL
```

10.177.1.666 CKR_PIN_INVALID

```
#define CKR_PIN_INVALID 0x000000A1UL
```

10.177.1.667 CKR_PIN_LEN_RANGE

```
#define CKR_PIN_LEN_RANGE 0x000000A2UL
```

10.177.1.668 CKR_PIN_LOCKED

```
#define CKR_PIN_LOCKED 0x000000A4UL
```

10.177.1.669 CKR_PIN_TOO_WEAK

```
#define CKR_PIN_TOO_WEAK 0x000001B8UL
```

10.177.1.670 CKR_PUBLIC_KEY_INVALID

```
#define CKR_PUBLIC_KEY_INVALID 0x000001B9UL
```

10.177.1.671 CKR_RANDOM_NO_RNG

```
#define CKR_RANDOM_NO_RNG 0x00000121UL
```

10.177.1.672 CKR_RANDOM_SEED_NOT_SUPPORTED

```
#define CKR_RANDOM_SEED_NOT_SUPPORTED 0x00000120UL
```

10.177.1.673 CKR_SAVED_STATE_INVALID

```
#define CKR_SAVED_STATE_INVALID 0x00000160UL
```

10.177.1.674 CKR_SESSION_CLOSED

```
#define CKR_SESSION_CLOSED 0x000000B0UL
```

10.177.1.675 CKR_SESSION_COUNT

```
#define CKR_SESSION_COUNT 0x000000B1UL
```

10.177.1.676 CKR_SESSION_EXISTS

```
#define CKR_SESSION_EXISTS 0x000000B6UL
```

10.177.1.677 CKR_SESSION_HANDLE_INVALID

```
#define CKR_SESSION_HANDLE_INVALID 0x000000B3UL
```

10.177.1.678 CKR_SESSION_PARALLEL_NOT_SUPPORTED

```
#define CKR_SESSION_PARALLEL_NOT_SUPPORTED 0x000000B4UL
```

10.177.1.679 CKR_SESSION_READ_ONLY

```
#define CKR_SESSION_READ_ONLY 0x000000B5UL
```

10.177.1.680 CKR_SESSION_READ_ONLY_EXISTS

```
#define CKR_SESSION_READ_ONLY_EXISTS 0x000000B7UL
```

10.177.1.681 CKR_SESSION_READ_WRITE_SO_EXISTS

```
#define CKR_SESSION_READ_WRITE_SO_EXISTS 0x000000B8UL
```

10.177.1.682 CKR_SIGNATURE_INVALID

```
#define CKR_SIGNATURE_INVALID 0x000000C0UL
```

10.177.1.683 CKR_SIGNATURE_LEN_RANGE

```
#define CKR_SIGNATURE_LEN_RANGE 0x000000C1UL
```

10.177.1.684 CKR_SLOT_ID_INVALID

```
#define CKR_SLOT_ID_INVALID 0x00000003UL
```

10.177.1.685 CKR_STATE_UNSAVEABLE

```
#define CKR_STATE_UNSAVEABLE 0x00000180UL
```

10.177.1.686 CKR_TEMPLATE_INCOMPLETE

```
#define CKR_TEMPLATE_INCOMPLETE 0x000000D0UL
```

10.177.1.687 CKR_TEMPLATE_INCONSISTENT

```
#define CKR_TEMPLATE_INCONSISTENT 0x000000D1UL
```

10.177.1.688 CKR_TOKEN_NOT_PRESENT

```
#define CKR_TOKEN_NOT_PRESENT 0x000000E0UL
```

10.177.1.689 CKR_TOKEN_NOT_RECOGNIZED

```
#define CKR_TOKEN_NOT_RECOGNIZED 0x000000E1UL
```

10.177.1.690 CKR_TOKEN_WRITE_PROTECTED

```
#define CKR_TOKEN_WRITE_PROTECTED 0x000000E2UL
```

10.177.1.691 CKR_UNWRAPPING_KEY_HANDLE_INVALID

```
#define CKR_UNWRAPPING_KEY_HANDLE_INVALID 0x000000F0UL
```

10.177.1.692 CKR_UNWRAPPING_KEY_SIZE_RANGE

```
#define CKR_UNWRAPPING_KEY_SIZE_RANGE 0x000000F1UL
```

10.177.1.693 CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT

```
#define CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT 0x000000F2UL
```

10.177.1.694 CKR_USER_ALREADY_LOGGED_IN

```
#define CKR_USER_ALREADY_LOGGED_IN 0x00000100UL
```

10.177.1.695 CKR_USER_ANOTHER_ALREADY_LOGGED_IN

```
#define CKR_USER_ANOTHER_ALREADY_LOGGED_IN 0x00000104UL
```

10.177.1.696 CKR_USER_NOT_LOGGED_IN

```
#define CKR_USER_NOT_LOGGED_IN 0x00000101UL
```

10.177.1.697 CKR_USER_PIN_NOT_INITIALIZED

```
#define CKR_USER_PIN_NOT_INITIALIZED 0x00000102UL
```

10.177.1.698 CKR_USER_TOO_MANY_TYPES

```
#define CKR_USER_TOO_MANY_TYPES 0x00000105UL
```

10.177.1.699 CKR_USER_TYPE_INVALID

```
#define CKR_USER_TYPE_INVALID 0x00000103UL
```

10.177.1.700 CKR_VENDOR_DEFINED

```
#define CKR_VENDOR_DEFINED 0x80000000UL
```

10.177.1.701 CKR_WRAPPED_KEY_INVALID

```
#define CKR_WRAPPED_KEY_INVALID 0x00000110UL
```

10.177.1.702 CKR_WRAPPED_KEY_LEN_RANGE

```
#define CKR_WRAPPED_KEY_LEN_RANGE 0x00000112UL
```

10.177.1.703 CKR_WRAPPING_KEY_HANDLE_INVALID

```
#define CKR_WRAPPING_KEY_HANDLE_INVALID 0x00000113UL
```

10.177.1.704 CKR_WRAPPING_KEY_SIZE_RANGE

```
#define CKR_WRAPPING_KEY_SIZE_RANGE 0x00000114UL
```

10.177.1.705 CKR_WRAPPING_KEY_TYPE_INCONSISTENT

```
#define CKR_WRAPPING_KEY_TYPE_INCONSISTENT 0x00000115UL
```

10.177.1.706 CKS_RO_PUBLIC_SESSION

```
#define CKS_RO_PUBLIC_SESSION 0UL
```

10.177.1.707 CKS_RO_USER_FUNCTIONS

```
#define CKS_RO_USER_FUNCTIONS 1UL
```

10.177.1.708 CKS_RW_PUBLIC_SESSION

```
#define CKS_RW_PUBLIC_SESSION 2UL
```

10.177.1.709 CKS_RW_SO_FUNCTIONS

```
#define CKS_RW_SO_FUNCTIONS 4UL
```

10.177.1.710 CKS_RW_USER_FUNCTIONS

```
#define CKS_RW_USER_FUNCTIONS 3UL
```

10.177.1.711 CKU_CONTEXT_SPECIFIC

```
#define CKU_CONTEXT_SPECIFIC 2UL
```

10.177.1.712 CKU_SO

```
#define CKU_SO 0UL
```

10.177.1.713 CKU_USER

```
#define CKU_USER 1UL
```

10.177.1.714 CKZ_DATA_SPECIFIED

```
#define CKZ_DATA_SPECIFIED 0x00000001UL
```

10.177.1.715 CKZ_SALT_SPECIFIED

```
#define CKZ_SALT_SPECIFIED 0x00000001UL
```

10.177.1.716 CRYPTOKI_VERSION_AMENDMENT

```
#define CRYPTOKI_VERSION_AMENDMENT 0
```

10.177.1.717 CRYPTOKI_VERSION_MAJOR

```
#define CRYPTOKI_VERSION_MAJOR 2
```

10.177.1.718 CRYPTOKI_VERSION_MINOR

```
#define CRYPTOKI_VERSION_MINOR 40
```

10.177.1.719 FALSE

```
#define FALSE CK_FALSE
```

10.177.1.720 TRUE

```
#define TRUE CK_TRUE
```

10.177.2 Typedef Documentation

10.177.2.1 CK_AES_CBC_ENCRYPT_DATA_PARAMS

```
typedef struct CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_AES_CBC_ENCRYPT_DATA_PARAMS
```

10.177.2.2 CK_AES_CBC_ENCRYPT_DATA_PARAMS_PTR

```
typedef CK_AES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_AES_CBC_ENCRYPT_DATA_PARAMS_PTR
```

10.177.2.3 CK_AES_CCM_PARAMS

```
typedef struct CK_AES_CCM_PARAMS CK_AES_CCM_PARAMS
```

10.177.2.4 CK_AES_CCM_PARAMS_PTR

```
typedef CK_AES_CCM_PARAMS CK_PTR CK_AES_CCM_PARAMS_PTR
```

10.177.2.5 CK_AES_CTR_PARAMS

```
typedef struct CK_AES_CTR_PARAMS CK_AES_CTR_PARAMS
```

10.177.2.6 CK_AES_CTR_PARAMS_PTR

```
typedef CK_AES_CTR_PARAMS CK_PTR CK_AES_CTR_PARAMS_PTR
```

10.177.2.7 CK_AES_GCM_PARAMS

```
typedef struct CK_AES_GCM_PARAMS CK_AES_GCM_PARAMS
```

10.177.2.8 CK_AES_GCM_PARAMS_PTR

```
typedef CK_AES_GCM_PARAMS CK_PTR CK_AES_GCM_PARAMS_PTR
```

10.177.2.9 CK_ARIA_CBC_ENCRYPT_DATA_PARAMS

```
typedef struct CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_ARIA_CBC_ENCRYPT_DATA_PARAMS
```

10.177.2.10 CK_ARIA_CBC_ENCRYPT_DATA_PARAMS_PTR

```
typedef CK_ARIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_ARIA_CBC_ENCRYPT_DATA_PARAMS_PTR
```

10.177.2.11 CK_ATTRIBUTE

```
typedef struct CK_ATTRIBUTE CK_ATTRIBUTE
```

10.177.2.12 CK_ATTRIBUTE_PTR

```
typedef CK_ATTRIBUTE CK_PTR CK_ATTRIBUTE_PTR
```

10.177.2.13 CK_ATTRIBUTE_TYPE

```
typedef CK_ULONG CK_ATTRIBUTE_TYPE
```

10.177.2.14 CK_BBOOL

```
typedef CK_BYTE CK_BBOOL
```

10.177.2.15 CK_BYTE

```
typedef unsigned char CK_BYTE
```

10.177.2.16 CK_BYTE_PTR

```
typedef CK_BYTE CK_PTR CK_BYTE_PTR
```

10.177.2.17 CK_C_INITIALIZE_ARGS

```
typedef struct CK_C_INITIALIZE_ARGS CK_C_INITIALIZE_ARGS
```

10.177.2.18 CK_C_INITIALIZE_ARGS_PTR

```
typedef CK_C_INITIALIZE_ARGS CK_PTR CK_C_INITIALIZE_ARGS_PTR
```

10.177.2.19 CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS

```
typedef struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS
```

10.177.2.20 CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS_PTR

```
typedef CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS_PTR
```

10.177.2.21 CK_CAMELLIA_CTR_PARAMS

```
typedef struct CK_CAMELLIA_CTR_PARAMS CK_CAMELLIA_CTR_PARAMS
```

10.177.2.22 CK_CAMELLIA_CTR_PARAMS_PTR

```
typedef CK_CAMELLIA_CTR_PARAMS CK_PTR CK_CAMELLIA_CTR_PARAMS_PTR
```

10.177.2.23 CK_CCM_PARAMS

```
typedef struct CK_CCM_PARAMS CK_CCM_PARAMS
```

10.177.2.24 CK_CCM_PARAMS_PTR

```
typedef CK_CCM_PARAMS CK_PTR CK_CCM_PARAMS_PTR
```

10.177.2.25 CK_CERTIFICATE_CATEGORY

```
typedef CK_ULONG CK_CERTIFICATE_CATEGORY
```

10.177.2.26 CK_CERTIFICATE_TYPE

```
typedef CK_ULONG CK_CERTIFICATE_TYPE
```

10.177.2.27 CK_CHAR

```
typedef CK_BYTE CK_CHAR
```

10.177.2.28 CK_CHAR_PTR

```
typedef CK_CHAR CK_PTR CK_CHAR_PTR
```

10.177.2.29 CK_CMS_SIG_PARAMS

```
typedef struct CK_CMS_SIG_PARAMS CK_CMS_SIG_PARAMS
```

10.177.2.30 CK_CMS_SIG_PARAMS_PTR

```
typedef CK_CMS_SIG_PARAMS CK_PTR CK_CMS_SIG_PARAMS_PTR
```

10.177.2.31 CK_DATE

```
typedef struct CK_DATE CK_DATE
```

10.177.2.32 CK_DES_CBC_ENCRYPT_DATA_PARAMS

```
typedef struct CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_DES_CBC_ENCRYPT_DATA_PARAMS
```

10.177.2.33 CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR

```
typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR
```

10.177.2.34 CK_DSA_PARAMETER_GEN_PARAM

```
typedef struct CK_DSA_PARAMETER_GEN_PARAM CK_DSA_PARAMETER_GEN_PARAM
```

10.177.2.35 CK_DSA_PARAMETER_GEN_PARAM_PTR

```
typedef CK_DSA_PARAMETER_GEN_PARAM CK_PTR CK_DSA_PARAMETER_GEN_PARAM_PTR
```

10.177.2.36 CK_EC_KDF_TYPE

```
typedef CK_ULONG CK_EC_KDF_TYPE
```

10.177.2.37 CK_ECDH1_DERIVE_PARAMS

```
typedef struct CK_ECDH1_DERIVE_PARAMS CK_ECDH1_DERIVE_PARAMS
```

10.177.2.38 CK_ECDH1_DERIVE_PARAMS_PTR

```
typedef CK_ECDH1_DERIVE_PARAMS CK_PTR CK_ECDH1_DERIVE_PARAMS_PTR
```

10.177.2.39 CK_ECDH2_DERIVE_PARAMS

```
typedef struct CK_ECDH2_DERIVE_PARAMS CK_ECDH2_DERIVE_PARAMS
```

10.177.2.40 CK_ECDH2_DERIVE_PARAMS_PTR

```
typedef CK_ECDH2_DERIVE_PARAMS CK_PTR CK_ECDH2_DERIVE_PARAMS_PTR
```

10.177.2.41 CK_ECDH_AES_KEY_WRAP_PARAMS

```
typedef struct CK_ECDH_AES_KEY_WRAP_PARAMS CK_ECDH_AES_KEY_WRAP_PARAMS
```

10.177.2.42 CK_ECDH_AES_KEY_WRAP_PARAMS_PTR

```
typedef CK_ECDH_AES_KEY_WRAP_PARAMS CK_PTR CK_ECDH_AES_KEY_WRAP_PARAMS_PTR
```

10.177.2.43 CK_ECMQV_DERIVE_PARAMS

```
typedef struct CK_ECMQV_DERIVE_PARAMS CK_ECMQV_DERIVE_PARAMS
```

10.177.2.44 CK_ECMQV_DERIVE_PARAMS_PTR

```
typedef CK_ECMQV_DERIVE_PARAMS CK_PTR CK_ECMQV_DERIVE_PARAMS_PTR
```

10.177.2.45 CK_EXTRACT_PARAMS

```
typedef CK_ULONG CK_EXTRACT_PARAMS
```

10.177.2.46 CK_EXTRACT_PARAMS_PTR

```
typedef CK_EXTRACT_PARAMS CK_PTR CK_EXTRACT_PARAMS_PTR
```

10.177.2.47 CK_FLAGS

```
typedef CK_ULONG CK_FLAGS
```

10.177.2.48 CK_FUNCTION_LIST

```
typedef struct CK_FUNCTION_LIST CK_FUNCTION_LIST
```

10.177.2.49 CK_FUNCTION_LIST_PTR

```
typedef CK_FUNCTION_LIST CK_PTR CK_FUNCTION_LIST_PTR
```

10.177.2.50 CK_FUNCTION_LIST_PTR_PTR

```
typedef CK_FUNCTION_LIST_PTR CK_PTR CK_FUNCTION_LIST_PTR_PTR
```

10.177.2.51 CK_GCM_PARAMS

```
typedef struct CK_GCM_PARAMS CK_GCM_PARAMS
```

10.177.2.52 CK_GCM_PARAMS_PTR

```
typedef CK_GCM_PARAMS CK_PTR CK_GCM_PARAMS_PTR
```

10.177.2.53 CK_GOSTR3410_DERIVE_PARAMS

```
typedef struct CK_GOSTR3410_DERIVE_PARAMS CK_GOSTR3410_DERIVE_PARAMS
```

10.177.2.54 CK_GOSTR3410_DERIVE_PARAMS_PTR

```
typedef CK_GOSTR3410_DERIVE_PARAMS CK_PTR CK_GOSTR3410_DERIVE_PARAMS_PTR
```

10.177.2.55 CK_GOSTR3410_KEY_WRAP_PARAMS

```
typedef struct CK_GOSTR3410_KEY_WRAP_PARAMS CK_GOSTR3410_KEY_WRAP_PARAMS
```

10.177.2.56 CK_GOSTR3410_KEY_WRAP_PARAMS_PTR

```
typedef CK_GOSTR3410_KEY_WRAP_PARAMS CK_PTR CK_GOSTR3410_KEY_WRAP_PARAMS_PTR
```

10.177.2.57 CK_HW_FEATURE_TYPE

```
typedef CK_ULONG CK_HW_FEATURE_TYPE
```

10.177.2.58 CK_INFO

```
typedef struct CK_INFO CK_INFO
```

10.177.2.59 CK_INFO_PTR

```
typedef CK_INFO CK_PTR CK_INFO_PTR
```

10.177.2.60 CK_JAVA_MIDP_SECURITY_DOMAIN

```
typedef CK_ULONG CK_JAVA_MIDP_SECURITY_DOMAIN
```

10.177.2.61 CK_KEA_DERIVE_PARAMS

```
typedef struct CK_KEA_DERIVE_PARAMS CK_KEA_DERIVE_PARAMS
```

10.177.2.62 CK_KEA_DERIVE_PARAMS_PTR

```
typedef CK_KEA_DERIVE_PARAMS CK_PTR CK_KEA_DERIVE_PARAMS_PTR
```

10.177.2.63 CK_KEY_DERIVATION_STRING_DATA

```
typedef struct CK_KEY_DERIVATION_STRING_DATA CK_KEY_DERIVATION_STRING_DATA
```

10.177.2.64 CK_KEY_DERIVATION_STRING_DATA_PTR

```
typedef CK_KEY_DERIVATION_STRING_DATA CK_PTR CK_KEY_DERIVATION_STRING_DATA_PTR
```

10.177.2.65 CK_KEY_TYPE

```
typedef CK_ULONG CK_KEY_TYPE
```

10.177.2.66 CK_KEY_WRAP_SET_OAEP_PARAMS

```
typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS CK_KEY_WRAP_SET_OAEP_PARAMS
```

10.177.2.67 CK_KEY_WRAP_SET_OAEP_PARAMS_PTR

```
typedef CK_KEY_WRAP_SET_OAEP_PARAMS CK_PTR CK_KEY_WRAP_SET_OAEP_PARAMS_PTR
```

10.177.2.68 CK_KIP_PARAMS

```
typedef struct CK_KIP_PARAMS CK_KIP_PARAMS
```

10.177.2.69 CK_KIP_PARAMS_PTR

```
typedef CK_KIP_PARAMS CK_PTR CK_KIP_PARAMS_PTR
```

10.177.2.70 CK_LONG

```
typedef long int CK_LONG
```

10.177.2.71 CK_MAC_GENERAL_PARAMS

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS
```

10.177.2.72 CK_MAC_GENERAL_PARAMS_PTR

```
typedef CK_MAC_GENERAL_PARAMS CK_PTR CK_MAC_GENERAL_PARAMS_PTR
```

10.177.2.73 CK_MECHANISM

```
typedef struct CK_MECHANISM CK_MECHANISM
```

10.177.2.74 CK_MECHANISM_INFO

```
typedef struct CK_MECHANISM_INFO CK_MECHANISM_INFO
```

10.177.2.75 CK_MECHANISM_INFO_PTR

```
typedef CK_MECHANISM_INFO CK_PTR CK_MECHANISM_INFO_PTR
```

10.177.2.76 CK_MECHANISM_PTR

```
typedef CK_MECHANISM CK_PTR CK_MECHANISM_PTR
```

10.177.2.77 CK_MECHANISM_TYPE

```
typedef CK_ULONG CK_MECHANISM_TYPE
```

10.177.2.78 CK_MECHANISM_TYPE_PTR

```
typedef CK_MECHANISM_TYPE CK_PTR CK_MECHANISM_TYPE_PTR
```

10.177.2.79 CK_NOTIFICATION

```
typedef CK_ULONG CK_NOTIFICATION
```

10.177.2.80 CK_OBJECT_CLASS

```
typedef CK_ULONG CK_OBJECT_CLASS
```

10.177.2.81 CK_OBJECT_CLASS_PTR

```
typedef CK_OBJECT_CLASS CK_PTR CK_OBJECT_CLASS_PTR
```

10.177.2.82 CK_OBJECT_HANDLE

```
typedef CK_ULONG CK_OBJECT_HANDLE
```

10.177.2.83 CK_OBJECT_HANDLE_PTR

```
typedef CK_OBJECT_HANDLE CK_PTR CK_OBJECT_HANDLE_PTR
```

10.177.2.84 CK_OTP_PARAM

```
typedef struct CK_OTP_PARAM CK_OTP_PARAM
```

10.177.2.85 CK_OTP_PARAM_PTR

```
typedef CK_OTP_PARAM CK_PTR CK_OTP_PARAM_PTR
```

10.177.2.86 CK_OTP_PARAM_TYPE

```
typedef CK_ULONG CK_OTP_PARAM_TYPE
```

10.177.2.87 CK_OTP_PARAMS

```
typedef struct CK_OTP_PARAMS CK_OTP_PARAMS
```

10.177.2.88 CK_OTP_PARAMS_PTR

```
typedef CK_OTP_PARAMS CK_PTR CK_OTP_PARAMS_PTR
```

10.177.2.89 CK_OTP_SIGNATURE_INFO

```
typedef struct CK_OTP_SIGNATURE_INFO CK_OTP_SIGNATURE_INFO
```

10.177.2.90 CK_OTP_SIGNATURE_INFO_PTR

```
typedef CK_OTP_SIGNATURE_INFO CK_PTR CK_OTP_SIGNATURE_INFO_PTR
```

10.177.2.91 CK_PARAM_TYPE

```
typedef CK_OTP_PARAM_TYPE CK_PARAM_TYPE
```

10.177.2.92 CK_PBE_PARAMS

```
typedef struct CK_PBE_PARAMS CK_PBE_PARAMS
```

10.177.2.93 CK_PBE_PARAMS_PTR

```
typedef CK_PBE_PARAMS CK_PTR CK_PBE_PARAMS_PTR
```

10.177.2.94 CK_PKCS5_PBKD2_PARAMS

```
typedef struct CK_PKCS5_PBKD2_PARAMS CK_PKCS5_PBKD2_PARAMS
```

10.177.2.95 CK_PKCS5_PBKD2_PARAMS2

```
typedef struct CK_PKCS5_PBKD2_PARAMS2 CK_PKCS5_PBKD2_PARAMS2
```

10.177.2.96 CK_PKCS5_PBKD2_PARAMS2_PTR

```
typedef CK_PKCS5_PBKD2_PARAMS2 CK_PTR CK_PKCS5_PBKD2_PARAMS2_PTR
```

10.177.2.97 CK_PKCS5_PBKD2_PARAMS_PTR

```
typedef CK_PKCS5_PBKD2_PARAMS CK_PTR CK_PKCS5_PBKD2_PARAMS_PTR
```

10.177.2.98 CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE

```
typedef CK_ULONG CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE
```

10.177.2.99 CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE_PTR

```
typedef CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE CK_PTR CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE_PTR
```

10.177.2.100 CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE

```
typedef CK_ULONG CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE
```

10.177.2.101 CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE_PTR

```
typedef CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE CK_PTR CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE_PTR
```

10.177.2.102 CK_RC2_CBC_PARAMS

```
typedef struct CK_RC2_CBC_PARAMS CK_RC2_CBC_PARAMS
```

10.177.2.103 CK_RC2_CBC_PARAMS_PTR

```
typedef CK_RC2_CBC_PARAMS CK_PTR CK_RC2_CBC_PARAMS_PTR
```

10.177.2.104 CK_RC2_MAC_GENERAL_PARAMS

```
typedef struct CK_RC2_MAC_GENERAL_PARAMS CK_RC2_MAC_GENERAL_PARAMS
```

10.177.2.105 CK_RC2_MAC_GENERAL_PARAMS_PTR

```
typedef CK_RC2_MAC_GENERAL_PARAMS CK_PTR CK_RC2_MAC_GENERAL_PARAMS_PTR
```

10.177.2.106 CK_RC2_PARAMS

```
typedef CK_ULONG CK_RC2_PARAMS
```

10.177.2.107 CK_RC2_PARAMS_PTR

```
typedef CK_RC2_PARAMS CK_PTR CK_RC2_PARAMS_PTR
```

10.177.2.108 CK_RC5_CBC_PARAMS

```
typedef struct CK_RC5_CBC_PARAMS CK_RC5_CBC_PARAMS
```

10.177.2.109 CK_RC5_CBC_PARAMS_PTR

```
typedef CK_RC5_CBC_PARAMS CK_PTR CK_RC5_CBC_PARAMS_PTR
```

10.177.2.110 CK_RC5_MAC_GENERAL_PARAMS

```
typedef struct CK_RC5_MAC_GENERAL_PARAMS CK_RC5_MAC_GENERAL_PARAMS
```

10.177.2.111 CK_RC5_MAC_GENERAL_PARAMS_PTR

```
typedef CK_RC5_MAC_GENERAL_PARAMS CK_PTR CK_RC5_MAC_GENERAL_PARAMS_PTR
```

10.177.2.112 CK_RC5_PARAMS

```
typedef struct CK_RC5_PARAMS CK_RC5_PARAMS
```

10.177.2.113 CK_RC5_PARAMS_PTR

```
typedef CK_RC5_PARAMS CK_PTR CK_RC5_PARAMS_PTR
```

10.177.2.114 CK_RSA_AES_KEY_WRAP_PARAMS

```
typedef struct CK_RSA_AES_KEY_WRAP_PARAMS CK_RSA_AES_KEY_WRAP_PARAMS
```

10.177.2.115 CK_RSA_AES_KEY_WRAP_PARAMS_PTR

```
typedef CK_RSA_AES_KEY_WRAP_PARAMS CK_PTR CK_RSA_AES_KEY_WRAP_PARAMS_PTR
```

10.177.2.116 CK_RSA_PKCS_MGF_TYPE

```
typedef CK_ULONG CK_RSA_PKCS_MGF_TYPE
```

10.177.2.117 CK_RSA_PKCS_MGF_TYPE_PTR

```
typedef CK_RSA_PKCS_MGF_TYPE CK_PTR CK_RSA_PKCS_MGF_TYPE_PTR
```

10.177.2.118 CK_RSA_PKCS_OAEP_PARAMS

```
typedef struct CK_RSA_PKCS_OAEP_PARAMS CK_RSA_PKCS_OAEP_PARAMS
```

10.177.2.119 CK_RSA_PKCS_OAEP_PARAMS_PTR

```
typedef CK_RSA_PKCS_OAEP_PARAMS CK_PTR CK_RSA_PKCS_OAEP_PARAMS_PTR
```

10.177.2.120 CK_RSA_PKCS_OAEP_SOURCE_TYPE

```
typedef CK_ULONG CK_RSA_PKCS_OAEP_SOURCE_TYPE
```

10.177.2.121 CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR

```
typedef CK_RSA_PKCS_OAEP_SOURCE_TYPE CK_PTR CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR
```

10.177.2.122 CK_RSA_PKCS_PSS_PARAMS

```
typedef struct CK_RSA_PKCS_PSS_PARAMS CK_RSA_PKCS_PSS_PARAMS
```

10.177.2.123 CK_RSA_PKCS_PSS_PARAMS_PTR

```
typedef CK_RSA_PKCS_PSS_PARAMS CK_PTR CK_RSA_PKCS_PSS_PARAMS_PTR
```

10.177.2.124 CK_RV

```
typedef CK_ULONG CK_RV
```

10.177.2.125 CK_SEED_CBC_ENCRYPT_DATA_PARAMS

```
typedef struct CK_SEED_CBC_ENCRYPT_DATA_PARAMS CK_SEED_CBC_ENCRYPT_DATA_PARAMS
```

10.177.2.126 CK_SEED_CBC_ENCRYPT_DATA_PARAMS_PTR

```
typedef CK_SEED_CBC_ENCRYPT_DATA_PARAMS CK_PTR CK_SEED_CBC_ENCRYPT_DATA_PARAMS_PTR
```

10.177.2.127 CK_SESSION_HANDLE

```
typedef CK_ULONG CK_SESSION_HANDLE
```

10.177.2.128 CK_SESSION_HANDLE_PTR

```
typedef CK_SESSION_HANDLE CK_PTR CK_SESSION_HANDLE_PTR
```

10.177.2.129 CK_SESSION_INFO

```
typedef struct CK_SESSION_INFO CK_SESSION_INFO
```

10.177.2.130 CK_SESSION_INFO_PTR

```
typedef CK_SESSION_INFO CK_PTR CK_SESSION_INFO_PTR
```

10.177.2.131 CK_SKIPJACK_PRIVATE_WRAP_PARAMS

```
typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS CK_SKIPJACK_PRIVATE_WRAP_PARAMS
```

10.177.2.132 CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR

```
typedef CK_SKIPJACK_PRIVATE_WRAP_PARAMS CK_PTR CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR
```

10.177.2.133 CK_SKIPJACK_RELAYX_PARAMS

```
typedef struct CK_SKIPJACK_RELAYX_PARAMS CK_SKIPJACK_RELAYX_PARAMS
```

10.177.2.134 CK_SKIPJACK_RELAYX_PARAMS_PTR

```
typedef CK_SKIPJACK_RELAYX_PARAMS CK_PTR CK_SKIPJACK_RELAYX_PARAMS_PTR
```

10.177.2.135 CK_SLOT_ID

```
typedef CK_ULONG CK_SLOT_ID
```

10.177.2.136 CK_SLOT_ID_PTR

```
typedef CK_SLOT_ID CK_PTR CK_SLOT_ID_PTR
```

10.177.2.137 CK_SLOT_INFO

```
typedef struct CK_SLOT_INFO CK_SLOT_INFO
```

10.177.2.138 CK_SLOT_INFO_PTR

```
typedef CK_SLOT_INFO CK_PTR CK_SLOT_INFO_PTR
```

10.177.2.139 CK_SSL3_KEY_MAT_OUT

```
typedef struct CK_SSL3_KEY_MAT_OUT CK_SSL3_KEY_MAT_OUT
```

10.177.2.140 CK_SSL3_KEY_MAT_OUT_PTR

```
typedef CK_SSL3_KEY_MAT_OUT CK_PTR CK_SSL3_KEY_MAT_OUT_PTR
```

10.177.2.141 CK_SSL3_KEY_MAT_PARAMS

```
typedef struct CK_SSL3_KEY_MAT_PARAMS CK_SSL3_KEY_MAT_PARAMS
```

10.177.2.142 CK_SSL3_KEY_MAT_PARAMS_PTR

```
typedef CK_SSL3_KEY_MAT_PARAMS CK_PTR CK_SSL3_KEY_MAT_PARAMS_PTR
```

10.177.2.143 CK_SSL3_MASTER_KEY_DERIVE_PARAMS

```
typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS CK_SSL3_MASTER_KEY_DERIVE_PARAMS
```

10.177.2.144 CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR

```
typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR
```

10.177.2.145 CK_SSL3_RANDOM_DATA

```
typedef struct CK_SSL3_RANDOM_DATA CK_SSL3_RANDOM_DATA
```

10.177.2.146 CK_STATE

```
typedef CK_ULONG CK_STATE
```

10.177.2.147 CK_TLS12_KEY_MAT_PARAMS

```
typedef struct CK_TLS12_KEY_MAT_PARAMS CK_TLS12_KEY_MAT_PARAMS
```

10.177.2.148 CK_TLS12_KEY_MAT_PARAMS_PTR

```
typedef CK_TLS12_KEY_MAT_PARAMS CK_PTR CK_TLS12_KEY_MAT_PARAMS_PTR
```

10.177.2.149 CK_TLS12_MASTER_KEY_DERIVE_PARAMS

```
typedef struct CK_TLS12_MASTER_KEY_DERIVE_PARAMS CK_TLS12_MASTER_KEY_DERIVE_PARAMS
```

10.177.2.150 CK_TLS12_MASTER_KEY_DERIVE_PARAMS_PTR

```
typedef CK_TLS12_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_TLS12_MASTER_KEY_DERIVE_PARAMS_PTR
```

10.177.2.151 CK_TLS_KDF_PARAMS

```
typedef struct CK_TLS_KDF_PARAMS CK_TLS_KDF_PARAMS
```

10.177.2.152 CK_TLS_KDF_PARAMS_PTR

```
typedef CK_TLS_KDF_PARAMS CK_PTR CK_TLS_KDF_PARAMS_PTR
```

10.177.2.153 CK_TLS_MAC_PARAMS

```
typedef struct CK_TLS_MAC_PARAMS CK_TLS_MAC_PARAMS
```

10.177.2.154 CK_TLS_MAC_PARAMS_PTR

```
typedef CK_TLS_MAC_PARAMS CK_PTR CK_TLS_MAC_PARAMS_PTR
```

10.177.2.155 CK_TLS_PRF_PARAMS

```
typedef struct CK_TLS_PRF_PARAMS CK_TLS_PRF_PARAMS
```

10.177.2.156 CK_TLS_PRF_PARAMS_PTR

```
typedef CK_TLS_PRF_PARAMS CK_PTR CK_TLS_PRF_PARAMS_PTR
```

10.177.2.157 CK_TOKEN_INFO

```
typedef struct CK_TOKEN_INFO CK_TOKEN_INFO
```

10.177.2.158 CK_TOKEN_INFO_PTR

```
typedef CK_TOKEN_INFO CK_PTR CK_TOKEN_INFO_PTR
```

10.177.2.159 CK_ULONG

```
typedef unsigned long int CK_ULONG
```

10.177.2.160 CK_ULONG_PTR

```
typedef CK_ULONG CK_PTR CK_ULONG_PTR
```

10.177.2.161 CK_USER_TYPE

```
typedef CK_ULONG CK_USER_TYPE
```

10.177.2.162 CK_UTF8CHAR

```
typedef CK_BYTE CK_UTF8CHAR
```

10.177.2.163 CK_UTF8CHAR_PTR

```
typedef CK_UTF8CHAR CK_PTR CK_UTF8CHAR_PTR
```

10.177.2.164 CK_VERSION

```
typedef struct CK_VERSION CK_VERSION
```

10.177.2.165 CK_VERSION_PTR

```
typedef CK_VERSION CK_PTR CK_VERSION_PTR
```

10.177.2.166 CK_VOID_PTR

```
typedef void CK_PTR CK_VOID_PTR
```

10.177.2.167 CK_VOID_PTR_PTR

```
typedef CK_VOID_PTR CK_PTR CK_VOID_PTR_PTR
```

10.177.2.168 CK_WTLS_KEY_MAT_OUT

```
typedef struct CK_WTLS_KEY_MAT_OUT CK_WTLS_KEY_MAT_OUT
```

10.177.2.169 CK_WTLS_KEY_MAT_OUT_PTR

```
typedef CK_WTLS_KEY_MAT_OUT CK_PTR CK_WTLS_KEY_MAT_OUT_PTR
```

10.177.2.170 CK_WTLS_KEY_MAT_PARAMS

```
typedef struct CK_WTLS_KEY_MAT_PARAMS CK_WTLS_KEY_MAT_PARAMS
```

10.177.2.171 CK_WTLS_KEY_MAT_PARAMS_PTR

```
typedef CK_WTLS_KEY_MAT_PARAMS CK_PTR CK_WTLS_KEY_MAT_PARAMS_PTR
```

10.177.2.172 CK_WTLS_MASTER_KEY_DERIVE_PARAMS

```
typedef struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS CK_WTLS_MASTER_KEY_DERIVE_PARAMS
```

10.177.2.173 CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR

```
typedef CK_WTLS_MASTER_KEY_DERIVE_PARAMS CK_PTR CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR
```

10.177.2.174 CK_WTLS_PRF_PARAMS

```
typedef struct CK_WTLS_PRF_PARAMS CK_WTLS_PRF_PARAMS
```

10.177.2.175 CK_WTLS_PRF_PARAMS_PTR

```
typedef CK_WTLS_PRF_PARAMS CK_PTR CK_WTLS_PRF_PARAMS_PTR
```

10.177.2.176 CK_WTLS_RANDOM_DATA

```
typedef struct CK_WTLS_RANDOM_DATA CK_WTLS_RANDOM_DATA
```

10.177.2.177 CK_WTLS_RANDOM_DATA_PTR

```
typedef CK_WTLS_RANDOM_DATA CK_PTR CK_WTLS_RANDOM_DATA_PTR
```

10.177.2.178 CK_X9_42_DH1_DERIVE_PARAMS

```
typedef struct CK_X9_42_DH1_DERIVE_PARAMS CK_X9_42_DH1_DERIVE_PARAMS
```

10.177.2.179 CK_X9_42_DH1_DERIVE_PARAMS_PTR

```
typedef struct CK_X9_42_DH1_DERIVE_PARAMS CK_PTR CK_X9_42_DH1_DERIVE_PARAMS_PTR
```

10.177.2.180 CK_X9_42_DH2_DERIVE_PARAMS

```
typedef struct CK_X9_42_DH2_DERIVE_PARAMS CK_X9_42_DH2_DERIVE_PARAMS
```

10.177.2.181 CK_X9_42_DH2_DERIVE_PARAMS_PTR

```
typedef CK_X9_42_DH2_DERIVE_PARAMS CK_PTR CK_X9_42_DH2_DERIVE_PARAMS_PTR
```

10.177.2.182 CK_X9_42_DH_KDF_TYPE

```
typedef CK_ULONG CK_X9_42_DH_KDF_TYPE
```

10.177.2.183 CK_X9_42_DH_KDF_TYPE_PTR

```
typedef CK_X9_42_DH_KDF_TYPE CK_PTR CK_X9_42_DH_KDF_TYPE_PTR
```

10.177.2.184 CK_X9_42_MQV_DERIVE_PARAMS

```
typedef struct CK_X9_42_MQV_DERIVE_PARAMS CK_X9_42_MQV_DERIVE_PARAMS
```

10.177.2.185 CK_X9_42_MQV_DERIVE_PARAMS_PTR

```
typedef CK_X9_42_MQV_DERIVE_PARAMS CK_PTR CK_X9_42_MQV_DERIVE_PARAMS_PTR
```

10.177.2.186 event

```
typedef CK_NOTIFICATION event
```

10.177.2.187 pApplication

```
typedef CK_NOTIFICATION CK_VOID_PTR pApplication
```

10.177.3 Function Documentation

10.177.3.1 CK_CALLBACK_FUNCTION() [1/5]

```
typedef CK_CALLBACK_FUNCTION (
            CK_RV ,
            CK_CREATEMUTEX  )
```

10.177.3.2 CK_CALLBACK_FUNCTION() [2/5]

```
typedef CK_CALLBACK_FUNCTION (
            CK_RV ,
            CK_DESTROYMUTEX  )
```

10.177.3.3 CK_CALLBACK_FUNCTION() [3/5]

```
typedef CK_CALLBACK_FUNCTION (
            CK_RV ,
            CK_LOCKMUTEX  )
```

10.177.3.4 CK_CALLBACK_FUNCTION() [4/5]

```
typedef CK_CALLBACK_FUNCTION (
            CK_RV ,
            CK_NOTIFY  )
```

10.177.3.5 CK_CALLBACK_FUNCTION() [5/5]

```
typedef CK_CALLBACK_FUNCTION (
            CK_RV ,
            CK_UNLOCKMUTEX  )
```

10.178 README.md File Reference

10.179 README.md File Reference

10.180 README.md File Reference

10.181 README.md File Reference

10.182 README.md File Reference

10.183 README.md File Reference

10.184 README.md File Reference

10.185 README.md File Reference

10.186 README.md File Reference

10.187 README.md File Reference

10.188 readme.md File Reference

10.189 secure_boot.c File Reference

Provides required APIs to manage secure boot under various scenarios.

```
#include <string.h>
#include "secure_boot.h"
#include "io_protection_key.h"
#include "basic/atca_basic.h"
```

Functions

- ATCA_STATUS secure_boot_process (void)

  Handles secure boot functionality through initialization, execution, and de-initialization.
- ATCA_STATUS bind_host_and_secure_element_with_io_protection (uint16_t slot)

  Binds host MCU and Secure element with IO protection key.

10.189.1 Detailed Description

Provides required APIs to manage secure boot under various scenarios.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.189.2 Function Documentation

10.189.2.1 bind_host_and_secure_element_with_io_protection()

```
ATCA_STATUS bind_host_and_secure_element_with_io_protection (
            uint16_t slot )
```

Binds host MCU and Secure element with IO protection key.

Parameters

in	*slot*	The slot number of IO protection Key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.189.2.2 secure_boot_process()

```
ATCA_STATUS secure_boot_process (
            void )
```

Handles secure boot functionality through initialization, execution, and de-initialization.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.190 secure_boot.h File Reference

Provides required APIs to manage secure boot under various scenarios.

```
#include "atca_status.h"
#include "secure_boot_memory.h"
#include "atca_command.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Data Structures

- struct secure_boot_config_bits
- struct secure_boot_parameters

Macros

- #define SECURE_BOOT_CONFIG_DISABLE 0
- #define SECURE_BOOT_CONFIG_FULL_BOTH 1
- #define SECURE_BOOT_CONFIG_FULL_SIGN 2
- #define SECURE_BOOT_CONFIG_FULL_DIG 3
- #define SECURE_BOOT_CONFIGURATION SECURE_BOOT_CONFIG_FULL_DIG
- #define SECURE_BOOT_DIGEST_ENCRYPT_ENABLED true
- #define SECURE_BOOT_UPGRADE_SUPPORT true

Functions

- ATCA_STATUS secure_boot_process (void)

    Handles secure boot functionality through initialization, execution, and de-initialization.
- ATCA_STATUS bind_host_and_secure_element_with_io_protection (uint16_t slot)

    Binds host MCU and Secure element with IO protection key.
- ATCA_STATUS host_generate_random_number (uint8_t ∗rand)

10.190.1 Detailed Description

Provides required APIs to manage secure boot under various scenarios.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.190.2 Macro Definition Documentation

10.190.2.1 SECURE_BOOT_CONFIG_DISABLE

```
#define SECURE_BOOT_CONFIG_DISABLE 0
```

10.190.2.2 SECURE_BOOT_CONFIG_FULL_BOTH

```
#define SECURE_BOOT_CONFIG_FULL_BOTH 1
```

10.190.2.3 SECURE_BOOT_CONFIG_FULL_DIG

```
#define SECURE_BOOT_CONFIG_FULL_DIG 3
```

10.190.2.4 SECURE_BOOT_CONFIG_FULL_SIGN

```
#define SECURE_BOOT_CONFIG_FULL_SIGN 2
```

10.190.2.5 SECURE_BOOT_CONFIGURATION

```
#define SECURE_BOOT_CONFIGURATION SECURE_BOOT_CONFIG_FULL_DIG
```

10.190.2.6 SECURE_BOOT_DIGEST_ENCRYPT_ENABLED

```
#define SECURE_BOOT_DIGEST_ENCRYPT_ENABLED true
```

10.190.2.7 SECURE_BOOT_UPGRADE_SUPPORT

```
#define SECURE_BOOT_UPGRADE_SUPPORT true
```

10.190.3 Function Documentation

10.190.3.1 bind_host_and_secure_element_with_io_protection()

```
ATCA_STATUS bind_host_and_secure_element_with_io_protection (
             uint16_t slot )
```

Binds host MCU and Secure element with IO protection key.

Parameters

in	*slot*	The slot number of IO protection Key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.190.3.2 host_generate_random_number()

```
ATCA_STATUS host_generate_random_number (
            uint8_t * rand )
```

10.190.3.3 secure_boot_process()

```
ATCA_STATUS secure_boot_process (
            void )
```

Handles secure boot functionality through initialization, execution, and de-initialization.

Returns

ATCA_SUCCESS on success, otherwise an error code.

10.191 secure_boot_memory.h File Reference

Provides interface to memory component for the secure boot.

```
#include "atca_status.h"
#include "atca_command.h"
```

Data Structures

- struct memory_parameters

Functions

- ATCA_STATUS secure_boot_init_memory (memory_parameters *memory_params)
- ATCA_STATUS secure_boot_read_memory (uint8_t *pu8_data, uint32_t *pu32_target_length)
- ATCA_STATUS secure_boot_write_memory (uint8_t *pu8_data, uint32_t *pu32_target_length)
- void secure_boot_deinit_memory (memory_parameters *memory_params)
- ATCA_STATUS secure_boot_mark_full_copy_completion (void)
- bool secure_boot_check_full_copy_completion (void)

10.191.1 Detailed Description

Provides interface to memory component for the secure boot.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.191.2 Function Documentation

10.191.2.1 secure_boot_check_full_copy_completion()

```
bool secure_boot_check_full_copy_completion (
            void )
```

10.191.2.2 secure_boot_deinit_memory()

```
void secure_boot_deinit_memory (
            memory_parameters * memory_params )
```

10.191.2.3 secure_boot_init_memory()

```
ATCA_STATUS secure_boot_init_memory (
            memory_parameters * memory_params )
```

10.191.2.4 secure_boot_mark_full_copy_completion()

```
ATCA_STATUS secure_boot_mark_full_copy_completion (
            void )
```

10.191.2.5 secure_boot_read_memory()

```
ATCA_STATUS secure_boot_read_memory (
            uint8_t * pu8_data,
            uint32_t * pu32_target_length )
```

10.191.2.6 secure_boot_write_memory()

```
ATCA_STATUS secure_boot_write_memory (
            uint8_t * pu8_data,
            uint32_t * pu32_target_length )
```

10.192 sha1_routines.c File Reference

Software implementation of the SHA1 algorithm.

```
#include "sha1_routines.h"
#include <string.h>
#include "atca_compiler.h"
```

Functions

- void CL_hashInit (CL_HashContext *ctx)
  Initialize context for performing SHA1 hash in software.
- void CL_hashUpdate (CL_HashContext *ctx, const uint8_t *src, int nbytes)
  Add arbitrary data to a SHA1 hash.
- void CL_hashFinal (CL_HashContext *ctx, uint8_t *dest)
  Complete the SHA1 hash in software and return the digest.
- void CL_hash (uint8_t *msg, int msgBytes, uint8_t *dest)
  Perform SHA1 hash of data in software.
- void shaEngine (uint32_t *buf, uint32_t *h)

10.192.1 Detailed Description

Software implementation of the SHA1 algorithm.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.192.2 Function Documentation

10.192.2.1 CL_hash()

```
void CL_hash (
            uint8_t * msg,
            int msgBytes,
            uint8_t * dest )
```

Perform SHA1 hash of data in software.

Parameters

in	*msg*	Data to be hashed
in	*msgBytes*	Data size in bytes
out	*dest*	Digest is returned here (20 bytes)

10.192.2.2 CL_hashFinal()

```
void CL_hashFinal (
            CL_HashContext * ctx,
            uint8_t * dest )
```

Complete the SHA1 hash in software and return the digest.

Parameters

in	*ctx*	Hash context
out	*dest*	Digest is returned here (20 bytes)

10.192.2.3 CL_hashInit()

```
void CL_hashInit (
            CL_HashContext * ctx )
```

Initialize context for performing SHA1 hash in software.

Parameters

in	*ctx*	Hash context

10.192.2.4 CL_hashUpdate()

```
void CL_hashUpdate (
            CL_HashContext * ctx,
            const uint8_t * src,
            int nbytes )
```

Add arbitrary data to a SHA1 hash.

Parameters

in	*ctx*	Hash context
in	*src*	Data to be added to the hash
in	*nbytes*	Data size in bytes

10.192.2.5 shaEngine()

```
void shaEngine (
            uint32_t * buf,
            uint32_t * h )
```

10.193 sha1_routines.h File Reference

Software implementation of the SHA1 algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
```

Data Structures

- struct CL_HashContext

Macros

- #define U8 uint8_t
- #define U16 uint16_t
- #define U32 uint32_t
- #define memcpy_P memmove
- #define strcpy_P strcpy
- #define _WDRESET()
- #define _NOP()
- #define leftRotate(x, n) (x) = (((x) << (n)) | ((x) >> (32 - (n))))

Functions

- void shaEngine (uint32_t *buf, uint32_t *h)
- void CL_hashInit (CL_HashContext *ctx)

    Initialize context for performing SHA1 hash in software.
- void CL_hashUpdate (CL_HashContext *ctx, const uint8_t *src, int nbytes)

    Add arbitrary data to a SHA1 hash.
- void CL_hashFinal (CL_HashContext *ctx, uint8_t *dest)

    Complete the SHA1 hash in software and return the digest.
- void CL_hash (uint8_t *msg, int msgBytes, uint8_t *dest)

    Perform SHA1 hash of data in software.

10.193.1 Detailed Description

Software implementation of the SHA1 algorithm.

Copyright

10.193.2 Macro Definition Documentation

10.193.2.1 _NOP

```
#define _NOP( )
```

10.193.2.2 _WDRESET

```
#define _WDRESET( )
```

10.193.2.3 leftRotate

```
#define leftRotate(
             x,
             n ) (x) = (((x) << (n)) | ((x) >> (32 - (n))))
```

10.193.2.4 memcpy_P

```
#define memcpy_P memmove
```

10.193.2.5 strcpy_P

```
#define strcpy_P strcpy
```

10.193.2.6 U16

```
#define U16 uint16_t
```

10.193.2.7 U32

```
#define U32 uint32_t
```

10.193.2.8 U8

```
#define U8 uint8_t
```

10.193.3 Function Documentation

10.193.3.1 CL_hash()

```
void CL_hash (
            uint8_t * msg,
            int msgBytes,
            uint8_t * dest )
```

Perform SHA1 hash of data in software.

Parameters

in	*msg*	Data to be hashed
in	*msgBytes*	Data size in bytes
out	*dest*	Digest is returned here (20 bytes)

10.193.3.2 CL_hashFinal()

```
void CL_hashFinal (
            CL_HashContext * ctx,
            uint8_t * dest )
```

Complete the SHA1 hash in software and return the digest.

Parameters

in	*ctx*	Hash context
out	*dest*	Digest is returned here (20 bytes)

10.193.3.3 CL_hashInit()

```
void CL_hashInit (
            CL_HashContext * ctx )
```

Initialize context for performing SHA1 hash in software.

Parameters

in	*ctx*	Hash context

10.193.3.4 CL_hashUpdate()

```
void CL_hashUpdate (
            CL_HashContext * ctx,
            const uint8_t * src,
            int nbytes )
```

Add arbitrary data to a SHA1 hash.

Parameters

in	*ctx*	Hash context
in	*src*	Data to be added to the hash
in	*nbytes*	Data size in bytes

10.193.3.5 shaEngine()

```
void shaEngine (
            uint32_t * buf,
            uint32_t * h )
```

10.194 sha2_routines.c File Reference

Software implementation of the SHA256 algorithm.

```
#include <string.h>
#include "sha2_routines.h"
#include "atca_compiler.h"
```

Macros

- #define rotate_right(value, places) ((value >> places) | (value << (32 - places)))

Functions

- void sw_sha256_init (sw_sha256_ctx *ctx)
  Intialize the software SHA256.
- void sw_sha256_update (sw_sha256_ctx *ctx, const uint8_t *msg, uint32_t msg_size)
  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- void sw_sha256_final (sw_sha256_ctx *ctx, uint8_t digest[(32)])
  completes the final SHA256 calculation and returns the final digest/hash
- void sw_sha256 (const uint8_t *message, unsigned int len, uint8_t digest[(32)])
  single call convenience function which computes Hash of given data using SHA256 software

10.194.1 Detailed Description

Software implementation of the SHA256 algorithm.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.194.2 Macro Definition Documentation

10.194.2.1 rotate_right

```
#define rotate_right(
            value,
            places ) ((value >> places) | (value << (32 - places)))
```

10.194.3 Function Documentation

10.194.3.1 sw_sha256()

```
void sw_sha256 (
            const uint8_t * message,
            unsigned int len,
            uint8_t digest[(32)] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*message*	pointer to stream of data to hash
in	*len*	size of data stream to hash
out	*digest*	result

10.194.3.2 sw_sha256_final()

```
void sw_sha256_final (
            sw_sha256_ctx * ctx,
            uint8_t digest[(32)] )
```

completes the final SHA256 calculation and returns the final digest/hash

Parameters

in	*ctx*	ptr to context data structure
out	*digest*	receives the computed digest of the SHA 256

10.194.3.3 sw_sha256_init()

```
void sw_sha256_init (
            sw_sha256_ctx * ctx )
```

Intialize the software SHA256.

Parameters

in	*ctx*	SHA256 hash context

10.194.3.4 sw_sha256_update()

```
void sw_sha256_update (
            sw_sha256_ctx * ctx,
            const uint8_t * msg,
            uint32_t msg_size )
```

updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software

Parameters

in	*ctx*	SHA256 hash context
in	*msg*	Raw blocks to be processed
in	*msg_size*	The size of the message passed

10.195 sha2_routines.h File Reference

Software implementation of the SHA256 algorithm.

```
#include <stdint.h>
```

Data Structures

- struct sw_sha256_ctx

Macros

- #define SHA256_DIGEST_SIZE (32)
- #define SHA256_BLOCK_SIZE (64)

Functions

- void sw_sha256_init (sw_sha256_ctx ∗ctx)

  Intialize the software SHA256.
- void sw_sha256_update (sw_sha256_ctx ∗ctx, const uint8_t ∗message, uint32_t len)

  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- void sw_sha256_final (sw_sha256_ctx ∗ctx, uint8_t digest[(32)])

  completes the final SHA256 calculation and returns the final digest/hash
- void sw_sha256 (const uint8_t ∗message, unsigned int len, uint8_t digest[(32)])

  single call convenience function which computes Hash of given data using SHA256 software

10.195.1 Detailed Description

Software implementation of the SHA256 algorithm.

Copyright

10.195.2 Macro Definition Documentation

10.195.2.1 SHA256_BLOCK_SIZE

```
#define SHA256_BLOCK_SIZE (64)
```

10.195.2.2 SHA256_DIGEST_SIZE

```
#define SHA256_DIGEST_SIZE (32)
```

10.195.3 Function Documentation

10.195.3.1 sw_sha256()

```
void sw_sha256 (
            const uint8_t * message,
            unsigned int len,
            uint8_t digest[(32)] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*message*	pointer to stream of data to hash
in	*len*	size of data stream to hash
out	*digest*	result

10.195.3.2 sw_sha256_final()

```
void sw_sha256_final (
            sw_sha256_ctx * ctx,
            uint8_t digest[(32)] )
```

completes the final SHA256 calculation and returns the final digest/hash

Parameters

in	*ctx*	ptr to context data structure
out	*digest*	receives the computed digest of the SHA 256

10.195.3.3 sw_sha256_init()

```
void sw_sha256_init (
            sw_sha256_ctx * ctx )
```

Intialize the software SHA256.

Parameters

in	*ctx*	SHA256 hash context

10.195.3.4 sw_sha256_update()

```
void sw_sha256_update (
            sw_sha256_ctx * ctx,
            const uint8_t * msg,
            uint32_t msg_size )
```

updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software

Parameters

in	*ctx*	SHA256 hash context
in	*msg*	Raw blocks to be processed
in	*msg_size*	The size of the message passed

10.196 swi_uart_samd21_asf.c File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

```
#include <stdlib.h>
#include <stdio.h>
#include "swi_uart_samd21_asf.h"
#include "atca_helpers.h"
```

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

    implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

implementation of SWI UART change mode.

- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

    discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

    HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.

- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

    HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Variables

- struct port_config pin_conf

10.196.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.197 swi_uart_samd21_asf.h File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct atcaSWImaster

    this is the hal_data for ATCA HAL for ASF SERCOM

Macros

- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90
- #define DEBUG_PIN_1 EXT2_PIN_5
- #define DEBUG_PIN_2 EXT2_PIN_6

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

    this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

    implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

    implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

    discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

    HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

    HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

10.197.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

10.198 swi_uart_start.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <peripheral_clk_config.h>
#include "swi_uart_start.h"
#include "atca_helpers.h"
```

Macros

- #define USART_BAUD_RATE(baud, sercom_freq) (65536 - ((65536 ∗ 16.0F ∗ baud) / sercom_freq))

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

10.198.1 Detailed Description

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.198.2 Macro Definition Documentation

10.198.2.1 USART_BAUD_RATE

```
#define USART_BAUD_RATE(
              baud,
              sercom_freq ) (65536 - ((65536 * 16.0F * baud) / sercom_freq))
```

10.199 swi_uart_start.h File Reference

```
#include <stdlib.h>
#include "atmel_start.h"
#include "cryptoauthlib.h"
```

Data Structures

- struct atcaSWImaster

  this is the hal_data for ATCA HAL for ASF SERCOM

Macros

- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 93

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

10.199.1 Detailed Description

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.200 symmetric_authentication.c File Reference

Contains API for performing the symmetric Authentication between the Host and the device.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
#include "symmetric_authentication.h"
```

Functions

- ATCA_STATUS symmetric_authenticate (uint8_t slot, const uint8_t *master_key, const uint8_t *rand_number)

    Function which does the authentication between the host and device.

10.200.1 Detailed Description

Contains API for performing the symmetric Authentication between the Host and the device.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.200.2 Function Documentation

10.200.2.1 symmetric_authenticate()

```
ATCA_STATUS symmetric_authenticate (
            uint8_t slot,
            const uint8_t * master_key,
            const uint8_t * rand_number )
```

Function which does the authentication between the host and device.

Parameters

in	*slot*	The slot number used for the symmetric authentication.
in	*master_key*	The master key used for the calculating the symmetric key.
in	*rand_number*	The 20 byte rand_number from the host.

Returns

ATCA_SUCCESS on successful authentication, otherwise an error code.

10.201 symmetric_authentication.h File Reference

Contains API for performing the symmetric Authentication between the Host and the device.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS symmetric_authenticate (uint8_t slot, const uint8_t *master_key, const uint8_t *rand_number)

    Function which does the authentication between the host and device.

10.201.1 Detailed Description

Contains API for performing the symmetric Authentication between the Host and the device.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.201.2 Function Documentation

10.201.2.1 symmetric_authenticate()

```
ATCA_STATUS symmetric_authenticate (
            uint8_t slot,
            const uint8_t * master_key,
            const uint8_t * rand_number )
```

Function which does the authentication between the host and device.

Parameters

in	*slot*	The slot number used for the symmetric authentication.
in	*master_key*	The master key used for the calculating the symmetric key.
in	*rand_number*	The 20 byte rand_number from the host.

Returns

ATCA_SUCCESS on successful authentication, otherwise an error code.

10.202 tflxtls_cert_def_4_device.c File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tflxtls_cert_template_4_device [500]
- const atcacert_cert_element_t g_tflxtls_cert_elements_4_device [ ]
- const atcacert_def_t g_tflxtls_cert_def_4_device

10.202.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.202.2 Variable Documentation

10.202.2.1 g_tflxtls_cert_elements_4_device

```
const atcacert_cert_element_t g_tflxtls_cert_elements_4_device[ ]
```

10.202.2.2 g_tflxtls_cert_template_4_device

```
const uint8_t g_tflxtls_cert_template_4_device[500]
```

10.203 tflxtls_cert_def_4_device.h File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

Variables

- const atcacert_def_t g_tflxtls_cert_def_4_device

10.203.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.204 tng_atca.c File Reference

TNG Helper Functions.

```
#include <string.h>
#include "cryptoauthlib.h"
#include "tng_atca.h"
#include "tnglora_cert_def_2_device.h"
#include "tnglora_cert_def_4_device.h"
#include "tngtls_cert_def_2_device.h"
#include "tngtls_cert_def_3_device.h"
#include "tflxtls_cert_def_4_device.h"
#include "atcacert/atcacert_def.h"
```

Data Structures

- struct tng_cert_map_element

Functions

- const atcacert_def_t * tng_map_get_device_cert_def (int index)

  Helper function to iterate through all trust cert definitions.
- ATCA_STATUS tng_get_device_cert_def (const atcacert_def_t **cert_def)

  Get the TNG device certificate definition.
- ATCA_STATUS tng_get_device_pubkey (uint8_t *public_key)

  Uses GenKey command to calculate the public key from the primary device public key.

10.204.1 Detailed Description

TNG Helper Functions.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.205 tng_atca.h File Reference

TNG Helper Functions.

```
#include "atca_basic.h"
#include "atcacert/atcacert_def.h"
```

Functions

- const atcacert_def_t ∗ tng_map_get_device_cert_def (int index)

  Helper function to iterate through all trust cert definitions.
- ATCA_STATUS tng_get_device_cert_def (const atcacert_def_t ∗∗cert_def)

  Get the TNG device certificate definition.
- ATCA_STATUS tng_get_device_pubkey (uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from the primary device public key.

10.205.1 Detailed Description

TNG Helper Functions.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.206 tng_atcacert_client.c File Reference

Client side certificate I/O functions for TNG devices.

```
#include "tng_atca.h"
#include "atcacert/atcacert_client.h"
#include "tng_atcacert_client.h"
#include "tngtls_cert_def_1_signer.h"
#include "tng_root_cert.h"
```

Functions

- int tng_atcacert_max_device_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t ∗cert, size_t ∗cert_size, const uint8_t ∗signer_cert)

  Reads the device certificate for a TNG device.
- int tng_atcacert_device_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t ∗cert, size_t ∗cert_size)

  Reads the signer certificate for a TNG device.
- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

  Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

  Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

  Gets the root public key.

10.206.1 Detailed Description

Client side certificate I/O functions for TNG devices.

Copyright

10.206.2 Function Documentation

10.206.2.1 tng_atcacert_device_public_key()

```
int tng_atcacert_device_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the device public key is used from this certificate. If set to NULL, the device public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.2 tng_atcacert_max_signer_cert_size()

```
int tng_atcacert_max_signer_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.3 tng_atcacert_read_device_cert()

```
int tng_atcacert_read_device_cert (
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t * signer_cert )
```

Reads the device certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.
in	*signer_cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.4 tng_atcacert_read_signer_cert()

```
int tng_atcacert_read_signer_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Reads the signer certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.5 tng_atcacert_root_cert()

```
int tng_atcacert_root_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Get the TNG root cert.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.6 tng_atcacert_root_cert_size()

```
int tng_atcacert_root_cert_size (
            size_t * cert_size )
```

Get the size of the TNG root cert.

Parameters

out	*cert_size*	Certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.7 tng_atcacert_root_public_key()

```
int tng_atcacert_root_public_key (
            uint8_t * public_key )
```

Gets the root public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.206.2.8 tng_atcacert_signer_public_key()

```
int tng_atcacert_signer_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the signer public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

10.207 tng_atcacert_client.h File Reference

Client side certificate I/O functions for TNG devices.

```
#include <stdint.h>
#include "atcacert/atcacert.h"
```

Functions

- int tng_atcacert_max_device_cert_size (size_t *max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t *cert, size_t *cert_size, const uint8_t *signer_cert)

  Reads the device certificate for a TNG device.
- int tng_atcacert_device_public_key (uint8_t *public_key, uint8_t *cert)

  Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t *max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t *cert, size_t *cert_size)

Reads the signer certificate for a TNG device.

- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

  Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

  Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

  Gets the root public key.

10.207.1 Detailed Description

Client side certificate I/O functions for TNG devices.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.208 tng_root_cert.c File Reference

TNG root certificate (DER)

```
#include <stdint.h>
#include <stddef.h>
```

Variables

- const uint8_t g_cryptoauth_root_ca_002_cert [501]
- const size_t g_cryptoauth_root_ca_002_cert_size = sizeof(g_cryptoauth_root_ca_002_cert)

10.208.1 Detailed Description

TNG root certificate (DER)

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.208.2 Variable Documentation

10.208.2.1 g_cryptoauth_root_ca_002_cert

```
const uint8_t g_cryptoauth_root_ca_002_cert[501]
```

10.208.2.2 g_cryptoauth_root_ca_002_cert_size

```
const size_t g_cryptoauth_root_ca_002_cert_size = sizeof(g_cryptoauth_root_ca_002_cert)
```

10.209 tng_root_cert.h File Reference

TNG root certificate (DER)

```
#include <stdint.h>
```

- #define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266
- const uint8_t g_cryptoauth_root_ca_002_cert []
- const size_t g_cryptoauth_root_ca_002_cert_size

10.209.1 Detailed Description

TNG root certificate (DER)

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.210 tnglora_cert_def_1_signer.c File Reference

TNG LORA signer certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tngtls_cert_template_1_signer []
- const atcacert_cert_element_t g_tngtls_cert_elements_1_signer []
- SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_1_signer

10.210.1 Detailed Description

TNG LORA signer certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.210.2 Variable Documentation

10.210.2.1 g_tnglora_cert_def_1_signer

```
SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_1_signer
```

10.210.2.2 g_tngtls_cert_elements_1_signer

```
const atcacert_cert_element_t g_tngtls_cert_elements_1_signer[ ]  [extern]
```

10.210.2.3 g_tngtls_cert_template_1_signer

```
const uint8_t g_tngtls_cert_template_1_signer[ ]  [extern]
```

10.211 tnglora_cert_def_1_signer.h File Reference

TNG LORA signer certificate definition.

```
#include "atcacert/atcacert_def.h"
```

Variables

- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_1_signer

10.211.1 Detailed Description

TNG LORA signer certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.212 tnglora_cert_def_2_device.c File Reference

TNG LORA device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_2_device.h"
#include "tngtls_cert_def_1_signer.h"
#include "tnglora_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tngtls_cert_template_2_device [ ]
- const atcacert_cert_element_t g_tngtls_cert_elements_2_device [ ]
- SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_2_device

10.212.1 Detailed Description

TNG LORA device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.212.2 Variable Documentation

10.212.2.1 g_tnglora_cert_def_2_device

```
SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_2_device
```

10.212.2.2 g_tngtls_cert_elements_2_device

```
const atcacert_cert_element_t g_tngtls_cert_elements_2_device[ ]  [extern]
```

10.212.2.3 g_tngtls_cert_template_2_device

```
const uint8_t g_tngtls_cert_template_2_device[] [extern]
```

10.213 tnglora_cert_def_2_device.h File Reference

TNG LORA device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

Variables

- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_2_device

10.213.1 Detailed Description

TNG LORA device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.214 tnglora_cert_def_4_device.c File Reference

TNG LORA device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tnglora_cert_def_4_device.h"
#include "tnglora_cert_def_1_signer.h"
```

Variables

- SHARED_LIB_EXPORT const uint8_t g_tnglora_cert_template_4_device [552]
- SHARED_LIB_EXPORT const atcacert_cert_element_t g_tnglora_cert_elements_4_device [ ]
- SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_4_device

10.214.1 Detailed Description

TNG LORA device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.214.2 Variable Documentation

10.214.2.1 g_tnglora_cert_def_4_device

```
SHARED_LIB_EXPORT const atcacert_def_t g_tnglora_cert_def_4_device
```

10.214.2.2 g_tnglora_cert_elements_4_device

```
SHARED_LIB_EXPORT const atcacert_cert_element_t g_tnglora_cert_elements_4_device[]
```

10.214.2.3 g_tnglora_cert_template_4_device

```
SHARED_LIB_EXPORT const uint8_t g_tnglora_cert_template_4_device[552]
```

10.215 tnglora_cert_def_4_device.h File Reference

TNG LORA device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNGLORA_CERT_TEMPLATE_4_DEVICE_SIZE 552
- ATCA_DLL const atcacert_def_t g_tnglora_cert_def_4_device

10.215.1 Detailed Description

TNG LORA device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.216 tngtls_cert_def_1_signer.c File Reference

TNG TLS signer certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_1_signer.h"
```

Variables

- SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_1_signer [520]
- SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_1_signer [ ]
- SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_1_signer

10.216.1 Detailed Description

TNG TLS signer certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.216.2 Variable Documentation

10.216.2.1 g_tngtls_cert_def_1_signer

```
SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_1_signer
```

10.216.2.2 g_tngtls_cert_elements_1_signer

```
SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_1_signer[ ]
```

Initial value:

```
= {
    {
        .id = "subject",
        .device_loc ={
            .zone       = DEVZONE_NONE,
        },
        .cert_loc ={
            .offset = 158,
            .count  = 81
        }
    }
}
```

10.216.2.3 g_tngtls_cert_template_1_signer

```
SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_1_signer[520]
```

10.217 tngtls_cert_def_1_signer.h File Reference

TNG TLS signer certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNGTLS_CERT_TEMPLATE_1_SIGNER_SIZE 520
- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_1_signer

10.217.1 Detailed Description

TNG TLS signer certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.218 tngtls_cert_def_2_device.c File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_2_device.h"
#include "tngtls_cert_def_1_signer.h"
```

Variables

- SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_2_device [505]
- SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_2_device [2]
- SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_2_device

10.218.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.218.2 Variable Documentation

10.218.2.1 g_tngtls_cert_def_2_device

```
SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_2_device
```

10.218.2.2 g_tngtls_cert_elements_2_device

```
SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_2_device[2]
```

10.218.2.3 g_tngtls_cert_template_2_device

```
SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_2_device[505]
```

10.219 tngtls_cert_def_2_device.h File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNGTLS_CERT_TEMPLATE_2_DEVICE_SIZE 505
- #define TNGTLS_CERT_ELEMENTS_2_DEVICE_COUNT 2
- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_2_device

10.219.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.220 tngtls_cert_def_3_device.c File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tngtls_cert_def_3_device.h"
#include "tngtls_cert_def_1_signer.h"
```

Variables

- SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_3_device [546]
- SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_3_device [ ]
- SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_3_device

10.220.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.220.2 Variable Documentation

10.220.2.1 g_tngtls_cert_def_3_device

```
SHARED_LIB_EXPORT const atcacert_def_t g_tngtls_cert_def_3_device
```

10.220.2.2 g_tngtls_cert_elements_3_device

```
SHARED_LIB_EXPORT const atcacert_cert_element_t g_tngtls_cert_elements_3_device[]
```

10.220.2.3 g_tngtls_cert_template_3_device

```
SHARED_LIB_EXPORT const uint8_t g_tngtls_cert_template_3_device[546]
```

10.221 tngtls_cert_def_3_device.h File Reference

TNG TLS device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNGTLS_CERT_TEMPLATE_3_DEVICE_SIZE 546
- ATCA_DLL const atcacert_def_t g_tngtls_cert_def_3_device

10.221.1 Detailed Description

TNG TLS device certificate definition.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

10.222 trust_pkcs11_config.c File Reference

PKCS11 Trust Platform Configuration.

```
#include "cryptoauthlib.h"
#include "pkcs11_config.h"
#include "pkcs11/pkcs11_object.h"
#include "pkcs11/pkcs11_slot.h"
```

10.222.1 Detailed Description

PKCS11 Trust Platform Configuration.

Copyright

(c) 2015-2020 Microchip Technology Inc. and its subsidiaries.

Index