Crypto 16BV1 Library

Microchip Libraries for Applications (MLA)

Table of Contents

Crypto 16BV1 Library

1 Crypto 16BV1 Library

1.1 Introduction

This library provides symmetric cryptographic encryption and decryption functionality for the Microchip family of microcontrollers equipped with a hardware cryptographic engine in a convenient C language interface.

Description

This library provides symmetric and asymmetric cryptographic encryption and decryption functionality for the Microchip family of microcontrollers equipped with a hardware cryptographic engine in a convenient C language interface. This crypto library provides support for the AES, and TDES algorithms.

AES, and TDES are symmetric block cipher algorithms, meaning they encrypt/decrypt fixed-length blocks of data and use the same key for encryption and decryption. To provide a complete model of security, these algorithms should be used with one of the provided block cipher modes of operation.

AES is one of the most widely used ciphers available today. It uses 128-, 192-, or 256-bit keys to encrypt 128-bit blocks. AES supports the Electronic Codebook (ECB), Cipher-Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR), and Galois/Counter Mode (GCM) modes of operation.

TDES (Triple DES), based on the DES cipher, is a precursor to AES, and is maintained as a standard to allow time for transition to AES. **TDES is not recommended for new designs.** TDES uses 56-bit DES keys (64-bits, including parity bits) to encrypt 64-bit blocks. TDES actually uses up to three distinct keys, depending on the keyring option that the user is using (hence the name, Triple DES). TDES supports the Electronic Codebook (ECB), Cipher-Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB) modes of operation.

1.2 Legal Information

This software distribution is controlled by the Legal Information at www.microchip.com/mla_license

1.3 Release Notes

Release notes for the current version of the Crypto module.

Description

CRYPTO 16BV1 Library Version : 0.01b

This is the first release of the library.

Tested with MPLAB XC16 v1.21.

1.4 Using the Library

Describes how to use the crypto 16bv1 hardware library.

Description

This topic describes the basic architecture of the crypto hardware 16bv1 library and provides information and examples on how to use it.

Interface Header File: block_cipher_16bv1.h

The interface to the crypto 16bv1 library is defined in the "block_cipher_16bv1.h" header file. Any C language source (.c) file that uses the crypto 16bv1 library should include "block_cipher_16bv1.h". Additional header files are provided for the block cipher modes of operation; the generic header file for this is "block_cipher_16bv1.h" and each specific mode has its own header, with the form "block_cipher_16bv1_xxx.h," where "xxx" is the mode (ecb, cbc, cfb, ofb, ctr, gcm).

1.4.1 Abstraction Model

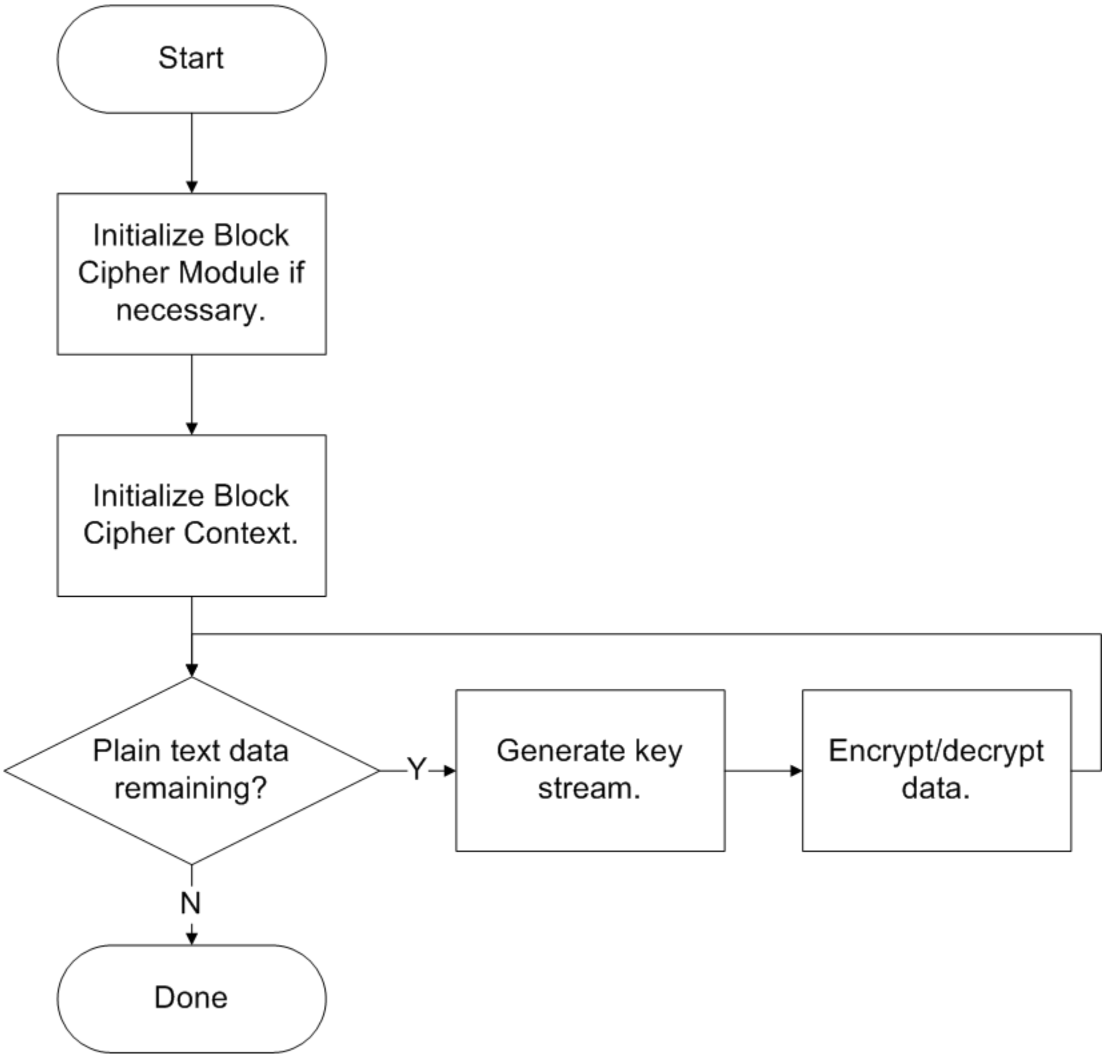
This library provides the low-level abstraction of the hardware crypto module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

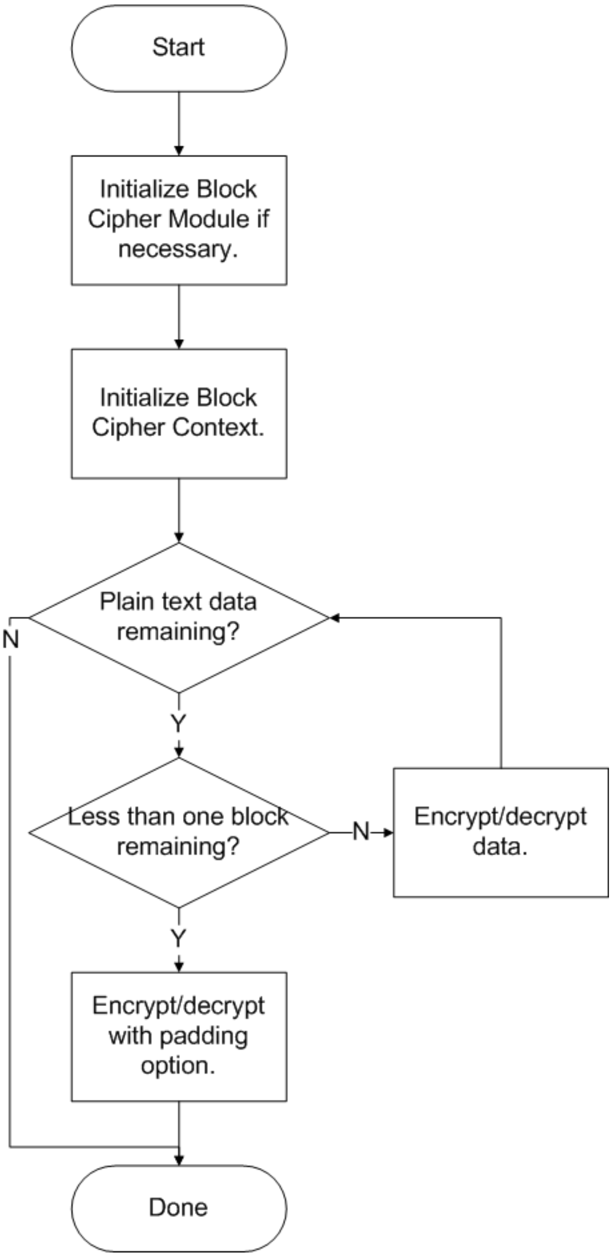
Non-authenticating Block Cipher Modules

Depending on the mode of operation used with a block cipher, the message may be padded, initialized with an initialization vector, or use feedback from previous encryption blocks to provide additional security. The currently available modes can be grouped into two categories: modes that use keystreams (OFB, CTR), and modes that do not (ECB, CBC, CFB). The keystream modes use initialization data (provided by feedback for OFB), but that data doesn't depend on the plaintext or ciphertext used for the previous encryption or decryption. For this reason, a keystream can be generated before the plaintext or ciphertext is available and then used to encrypt/decrypt a variable-length block of text when it becomes available. The other modes required whole blocks of data before they can be encrypted/decrypted.

Block Cipher Mode Module Software Abstraction Block Diagram (Keystream modes)


Start
Initialize Block Cipher Module if necessary.
Initialize Block Cipher Context.
Plain text data remaining?
Y
Generate key stream.
Encrypt/decrypt data.
N
Done

Block Cipher Mode Module Software Abstraction Block Diagram (Non-keystream modes)

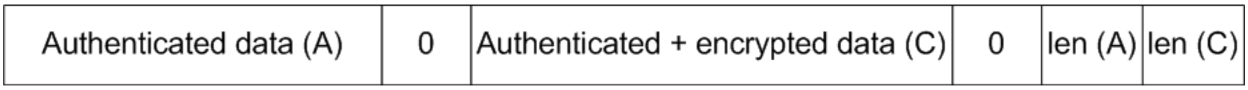

Start
Initialize Block Cipher Module if necessary.
Initialize Block Cipher Context.
Plain text data remaining?
N
Y
Less than one block remaining?
N
Encrypt/decrypt data.
Y
Encrypt/decrypt with padding option.
Done

Authenticating Block Cipher Modes

Galois/Counter Mode (GCM) is a special case. It provides encryption/decryption and authentication for a set of data. The encryption/decryption uses operations that are equivalent to counter mode, but the authentication mechanism operates on whole blocks of data. For this reason, GCM uses keystreams to encrypt/decrypt data, but must also be padded at the end of a set of encryptions/decryptions to generate an authentication tag. GCM can also authenticate a set of data that will not be encrypted. For example, if you have a packet of data with a header and a payload, you could use GCM to authenticate the header and payload with one authentication tag, but only encrypt the payload.

GCM operates on data in a specific order. First, data that is to be authenticated but not encrypted/decrypted is processed. If necessary this data is padded to one block size. For an encryption, the plaintext is then encrypted and the resulting ciphertext is authenticated. For a decryption, the ciphertext is authenticated, and then decrypted into a plaintext. The authenticated ciphertext is also padded. Finally, the lengths of the non-encrypted/decrypted data and ciphertext are authenticated, and an authentication tag is generated.

GCM Authenticated Data Organization



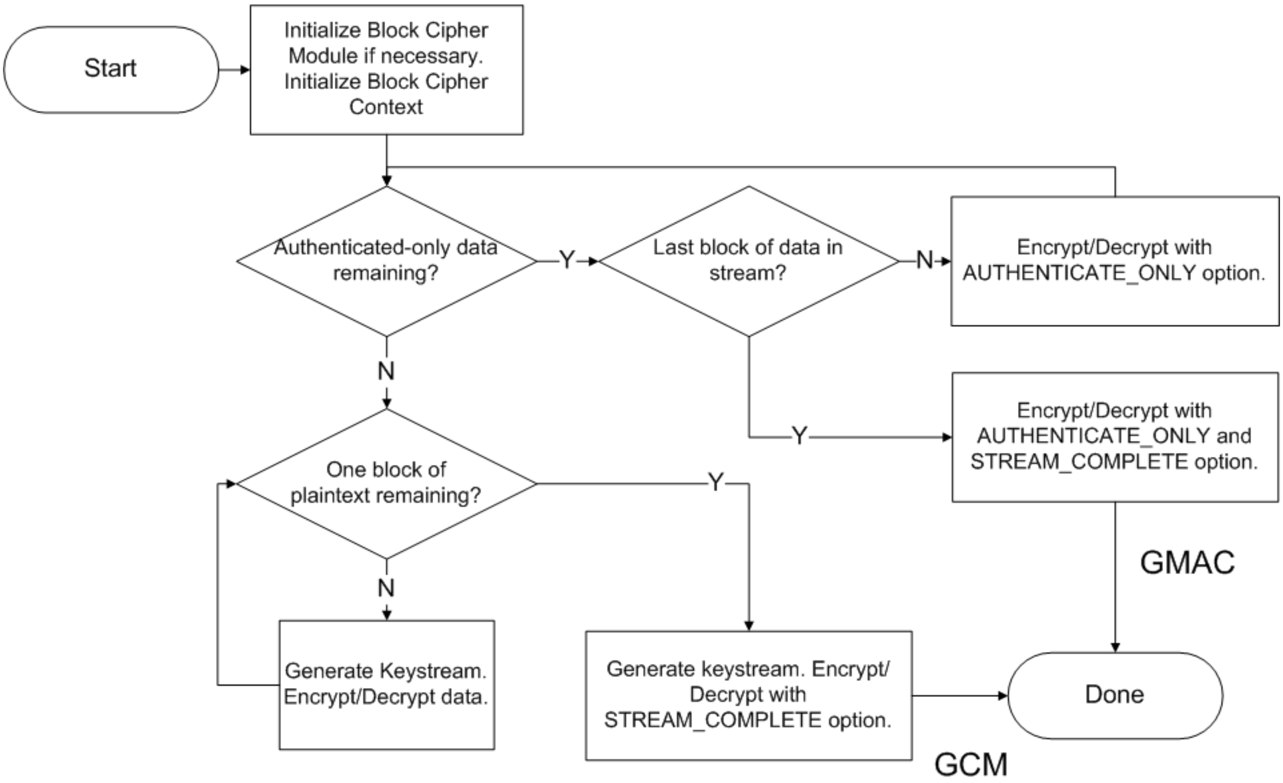
Authenticated data (A)	0	Authenticated + encrypted data (C)	0	len (A)	len (C)

The GCM module will take care of padding automatically, as long as the user specifies which operation should be performed on the data. When the user first calls BLOCK_CIPHER_GCM_Encrypt or BLOCK_CIPHER_GCM_Decrypt, he or she can optionally specify one of the options as BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY. This will indicate to the GCM module that the data being passed in should be authenticated, but not encrypted or decrypted. If this option is specified, the user does not need to specify an output buffer (the cipherText parameter for Encrypt, or the plainText parameter for Decrypt). Once the user has passed in all data that must be authenticated but not encrypted/decrypted, they can call the Encrypt or Decrypt function without the AUTHENTICATE_ONLY option. This will automatically generate zero-padding for the block of non-encrypted data.

If the user calls the Encrypt or Decrypt function without the AUTHENTICATE_ONLY option, any data they pass in to that call (and every subsequent call) will be both authenticated *and* encrypted or decrypted. Once the user is finished authenticating/encrypting/decrypting data, he or she will call the Encrypt or Decrypt function with the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option. This will indicate to the GCM module that all encryption and decryption has been completed, and it will pad the encrypted data with zeros (and with the lengths of the authenticated-only and the encrypted data) and calculate the final authentication tag. If the data is being encrypted, this tag will be returned to the user. If the data is being decrypted, this tag will be compared to a tag provided by the user and an error will be returned in the event of a mismatch.

Note that the user doesn't necessarily need to provide data to encrypt/decrypt. If the user only provides data with the BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY option, and specifies BLOCK_CIPHER_OPTION_STREAM_COMPLETE on the last block of authenticated data, an authentication tag will be produced, but there will be no resultant cipherText or plainText. This is known as a Galois Message Authentication Code (GMAC).

GCM/GMAC Software Abstraction Block Diagram


Start
Initialize Block Cipher Module if necessary. Initialize Block Cipher Context
Authenticated-only data remaining?
Y
Last block of data in stream?
N
Encrypt/Decrypt with AUTHENTICATE_ONLY option.
N
Y
Encrypt/Decrypt with AUTHENTICATE_ONLY and STREAM_COMPLETE option.
One block of plaintext remaining?
Y
GMAC
N
Generate Keystream. Encrypt/Decrypt data.
Generate keystream. Encrypt/ Decrypt with STREAM_COMPLETE option.
Done
GCM

1.4.2 Library Overview

Provides an overview of the crypto module.

Description

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the crypto module.

Library Interface Section	Description
Block Cipher Modes	Describes the API used by the general block cipher modes of operation module.

The block cipher modes of operation are designed to be used with the AES, and TDES algorithms. These block cipher modes do not provide a complete model of security without a mode of operation.

The AES, and TDES algorithms are implemented by the hardware engine directly for any part with a hardware cryptographic engine. When a block mode is being initialized the user selects AES or TDES by passing CRYPTO_16BV1_ALGORITHM_AES or CRYPTO_16BV1_ALGORITHM_TDES as the BLOCK_CIPHER_FunctionEncrypt parameter.

1.4.3 How the Library Works

Describes how the library works.

Description

Describes how the library works.

1.4.3.1 Block Ciphers

Describes how the block ciphers work, and how to use them with the block cipher modes of operation.

Description

Describes how the block ciphers work, and how to use them with the block cipher modes of operation.

1.4.3.1.1 Modes of Operation

Describes how the block cipher modes of operation work with each block cipher.

Description

General Functionality

Each mode of operation in the block cipher mode module is used with a specific cipher algorithm (e.g. AES). Before initializing the specific block cipher mode, the user must initialize the hardware module by calling BLOCK_CIPHER_16BV1_Initialize. The user must then use the block cipher's parameters and functions to initialize a block cipher mode context that will be used for the encryption/decryption. To do this, the user will initialize several parameters for the block cipher module:

- An uninitialized handle must be obtained from the hardware module by calling BLOCK_CIPHER_16BV1_Open
- A BLOCK_CIPHER_16BV1_[mode]_CONTEXT type data struct must be declared by the user
- The algorithm must be selected between CRYPTO_16BV1_ALGORITHM_AES or CRYPTO_16BV1_ALGORITHM_TDES

- The block size of the block cipher you are using. The AES, and TDES modules included with this cryptographic library define macros that indicate their block size (AES_BLOCK_SIZE, TDES_BLOCK_SIZE).
- A key must be specified by the user. including type mode and location as outlined by the types in block_cipher_16bv1.h
- Any initialization data needed by the mode you are using.

ECB

Using the ECB mode of operation is essentially the same as not using a mode of operation. This mode will encrypt blocks of data individually, without providing feedback from previous encryptions. **This mode does not provide sufficient security for use with cryptographic operations.** This mode will manage encryption/decryption of multiple blocks, cache data to be encrypted/decrypted if there is not enough to comprise a full block, and add padding at the end of a plain text based on user-specified options.

CBC

The Cipher-block Chaining (CBC) mode of operation uses an initialization vector and information from previous block encryptions to provide additional security.

Before the first encryption, the initialization vector is exclusive or'd (xor'd) with the first block of plaintext. After each encryption, the resulting block of ciphertext is xor'd with the next block of plaintext being encrypted.

When decrypting this message, the IV is xor'd with the first block of decrypted ciphertext to recover the first block of plaintext, the first block of ciphertext is xor'd with the second block of decrypted ciphertext, and so on.

CFB

Like the CBC mode, the Cipher Feedback (CFB) mode of operation uses an initialization vector and propagates information from en/decryptions to subsequent en/decryptions.

In CFB, the initialization vector is encrypted first, then the resulting value is xor'd with the first block of the plaintext to produce the first block of ciphertext. The first ciphertext is then encrypted, the resulting value is xor'd with the second block of plaintext to produce the second block of ciphertext, and so on.

When decrypting, the IV is encrypted again. The resulting value is xor'd with the ciphertext to produce the plaintext, and then the ciphertext is encrypted and xor'd with the next block of ciphertext to produce the second block of plaintext. This process continues until the entire message has been decrypted.

OFB

The Output Feedback (OFB) mode is the same as the CFB mode, except the data being encrypted for the subsequent encryptions is simply the result of the previous encryption instead of the result of the previous encryption xor'd with the plaintext. Note that the result of the encryption is still xor'd with the plaintext to produce the ciphertext; the value is just propagated to the next block encryption before this happens.

Since you don't need to have the plaintext before determining the encrypted values to xor with it, you can pre-generate a keystream for OFB as soon as you get the Initialization Vector and Key, and then use it to encrypt the plaintext when it becomes available. Also, since you can simply xor your keystream with a non-specific amount of plaintext, OFB is effectively a stream cipher, not a block cipher (thought you will still use the block cipher to generate the keystream).

CTR

The Counter (CTR) mode encrypts blocks that contain a counter to generate a keystream. This keystream is then xor'd with the plaintext to produce the ciphertext.

Usually the counter blocks are combined with an Initialization Vector (a security nonce) to provide additional security. In most cases the counter simply is incremented after each block is encrypted/decrypted, but any operation could be applied to the counter as long as the values of the counter didn't repeat frequently. CTR mode combines the advantages of ECB (blocks are encrypted/decrypted without need for information from previous operations, which allows encryptions to be run in parallel) with the advantages of OFB (keystreams can be generated before all of the plaintext is available).

GCM

The Galois/Counter Mode (GCM) is essentially the same as the counter mode for purposes of encryption and decryption. The difference is that GCM will also provide authentication functionality. GCM will use an initialization vector to generate an initial counter. That counter will be used with CTR-mode encryption to produce a ciphertext. The GCM will apply a hashing function to the ciphertext, a user-specified amount of non-encrypted data, and some padding data to produce an output. This hashed value will then be encrypted with the initial counter to produce an authentication tag. See the Abstraction Model topic for more information on how the authentication tag is constructed.

GCM provides several requirements and methods for constructing an initialization vector. In practice, the easiest way to create an acceptable Initialization Vector is to pass a 96-bit random number generated by an approved random bit generator with a sufficient security strength into the BLOCK_CIPHER_GCM_Initialize function. See section 8.2 in the GCM specification (NIST SP-800-32D) for more information.

1.4.3.1.2 AES

Describes how the AES module works.

Description

The AES module should be used with a block cipher mode of operation (see the block cipher modes of operation section for more information). For AES, the block cipher mode module's BLOCK_CIPHER_16BV1_[mode]_Initialize functions should be initialized with the CRYPTO_16BV1_ALGORITHM_AES parameter and the AES_BLOCK_SIZE block size macro. If an initialization vector or nonce/counter is required by the block cipher mode being used, it should be 16 bytes long (one block length).

If a software only key is being provided a pointer to the AES key is passed into the block cipher mode module's BLOCK_CIPHER_16BV1_[mode]_Initialize function. If an OTP key is being used the key should be selected based on its specific position in hardware CRYPTO_KEY_HARDWARE_OTP_[x], and matching the selection of key mode with the mode of key stored in OTP memory.

1.4.3.1.3 TDES

Describes how the TDES module works.

Description

The TDES module should be used with a block cipher mode of operation (see the block cipher modes of operation section for more information). For TDES, the block cipher mode module's BLOCK_CIPHER_16BV1_[mode]_Initialize functions should be initialized with the CRYPTO_16BV1_ALGORITHM_TDES parameter, and the TDES_BLOCK_SIZE block size macro. If an initialization vector or nonce/counter is required by the block cipher mode being used, it should be 8 bytes long (one block length).

TDES uses up to 3 64-bit DES keys, depending on the keyring option being used. In keyring option 1, all three keys will be distinct. This provides the most security. In keyring option 2, the first and third key are the same. This provides more security than the DES algorithm that TDES is based on. Keyring option 3 uses the same key three times. It is functionally equivalent to DES, and is provided for backwards compatibility; it should not be used in new applications.

If a software only key is being provided a pointer to the TDES key is passed into the block cipher mode module's BLOCK_CIPHER_16BV1_[mode]_Initialize function. If an OTP key is being used the key should be selected based on its specific position in hardware CRYPTO_KEY_HARDWARE_OTP_[x], and matching the selection of key mode with the mode of key stored in OTP memory.

1.5 Configuring the Library

Describes the crypto library configuration.

Macros

Name	Description
CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE	Defines the maximum block size of the hardware module
CRYPTO_CONFIG_16BV1_BLOCK_HANDLE_MAXIMUM	Defines the maximum number of user handles allowed to be open on the hardware module.

Description

The configuration of the crypto module is based on the file crypto_16bv1_config.h. This file (or the definitions it describes) must be included in a header named system_config.h, which will be included directly by the source files.

The crypto_16bv1_config.h header file contains the configuration selection for this cryptographic module, including the general configuration for block cipher modes. Based on the selections made, the crypto module will support or not support selected features. These configuration settings will apply to all instances of the crypto module.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build.

1.5.1 CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE Macro

File

crypto_16bv1_config_template.h

Syntax

```
#define CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE 16ul       // Defines the maximum block size of
the hardware module
```

Description

Defines the maximum block size of the hardware module

1.5.2 CRYPTO_CONFIG_16BV1_BLOCK_HANDLE_MAXIMUM Macro

File

crypto_16bv1_config_template.h

Syntax

```
#define CRYPTO_CONFIG_16BV1_BLOCK_HANDLE_MAXIMUM 10u     // Defines the maximum number of
user handles allowed to be open on the hardware module.
```

Description

Defines the maximum number of user handles allowed to be open on the hardware module.

1.6 Building the Library

Describes source files used by the crypto library.

Description

This section lists the files that are available in the \src of the crypto library. It lists which files need to be included in the build based on either a hardware feature present on the board or configuration option selected by the system.

1.6.1 Block Cipher Modes

This section describes the source files that must be included when building the block cipher modes of operation.

Description

This section describes the source files that must be included when building the AES module.

These files are located in the `crypto_16bv1/src/block_cipher` directory.

File	Description	Conditions
block_cipher_16bv1.c	Contains general purpose functions.	Must be included.
block_cipher_16bv1_cbc.c	Contains functions used for the CBC mode of operation.	Must be included when using CBC mode.
block_cipher_16bv1_cfb.c	Contains functions used for the CFB mode of operation.	Must be included when using CFB mode.
block_cipher_16bv1_cfb1.c	Contains functions used for the CFB1 mode of operation.	Must be included when using CFB1 mode.
block_cipher_16bv1_cfb8.c	Contains functions used for the CFB8 mode of operation.	Must be included when using CFB8 mode.
block_cipher_16bv1_ofb.c	Contains functions used for the OFB mode of operation.	Must be included when using OFB mode.
block_cipher_16bv1_ctr.c	Contains functions used for the CTR mode of operation.	Must be included when using CTR mode.
block_cipher_16bv1_ecb.c	Contains functions used for the ECB mode of operation.	Must be included when using ECB mode.
block_cipher_16bv1_gcm.c	Contains functions used for the GCM mode of operation.	Must be included when using GCM mode.

1.7 Library Interface

This section describes the Application Programming Interface (API) functions of the crypto module.

Refer to each section for a detailed description.

Description

This section describes the Application Programming Interface (API) functions of the crypto module.

Refer to each section for a detailed description.

1.7.1 Block Cipher Modes

Describes the functions and structures used to interface to this module.

Modules

Name	Description
General Functionality	Describes general functionality used by the block cipher mode module.
ECB	Describes functionality specific to the Electronic Codebook (ECB) block cipher mode of operation.
CBC	Describes functionality specific to the Cipher-Block Chaining (CBC) block cipher mode of operation.
CFB	Describes functionality specific to the Cipher Feedback (CFB) block cipher mode of operation.
OFB	Describes functionality specific to the Output Feedback (OFB) block cipher mode of operation.
CTR	Describes functionality specific to the Counter (CTR) block cipher mode of operation.
GCM	Describes functionality specific to the Galois/Counter Mode (GCM) block cipher mode of operation.

Description

Describes the functions and structures used to interface to this module.

1.7.1.1 General Functionality

Describes general functionality used by the block cipher mode module.

Enumerations

Name	Description
BLOCK_CIPHER_ERRORS	Enumeration defining potential errors the can occur when using a block cipher mode of operation. Modes that do not use keystreams will not generate errors.
BLOCK_CIPHER_STATE	This is type BLOCK_CIPHER_STATE.
CRYPTO_KEY_MODE	This is type CRYPTO_KEY_MODE.
CRYPTO_KEY_TYPE	Enumeration defining different key types

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_Close	Closes an opened client
◆	BLOCK_CIPHER_16BV1_Deinitialize	Deinitializes the instance of the block cipher module
◆	BLOCK_CIPHER_16BV1_GetState	Returns the current state of the client handle
◆	BLOCK_CIPHER_16BV1_Initialize	Initializes the data for the instance of the block cipher module.
◆	BLOCK_CIPHER_16BV1_Open	Opens a new client for the device instance.
◆	BLOCK_CIPHER_16BV1_Tasks	Runs all tasks necessary for processing a request

Macros

Name	Description
_BLOCK_CIPHER_16BV1_H	This is macro _BLOCK_CIPHER_16BV1_H.
BLOCK_CIPHER_HANDLE_INVALID	This is macro BLOCK_CIPHER_HANDLE_INVALID.
BLOCK_CIPHER_INDEX	Map of the default drive index to drive index 0
BLOCK_CIPHER_INDEX_0	Definition for a single drive index for the hardware AES/TDES module
BLOCK_CIPHER_INDEX_COUNT	Number of drive indicies for this module
CRYPTO_16BV1_ALGORITHM_AES	This is macro CRYPTO_16BV1_ALGORITHM_AES.
CRYPTO_16BV1_ALGORITHM_TDES	This is macro CRYPTO_16BV1_ALGORITHM_TDES.

Types

Name	Description
BLOCK_CIPHER_FunctionEncrypt	Function pointer for a block cipher's encryption function. When using the block cipher modes of operation module, you will configure it to use the encrypt function of the block cipher module that you are using with a pointer to that block cipher's encrypt function. None
BLOCK_CIPHER_FunctionDecrypt	Function pointer for a block cipher's decryption function. When using the block cipher modes of operation module, you will configure it to use the decrypt function of the block cipher module that you are using with a pointer to that block cipher's encrypt function. None
BLOCK_CIPHER_HANDLE	Block cipher handle type

Description

Describes general functionality used by the block cipher mode module.

1.7.1.1.1 Options

Describes general options that can be selected when encrypting/decrypting a message.

Macros

Name	Description
BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY	This option is used to pass data that will be authenticated but not encrypted into an authenticating block cipher mode function.
BLOCK_CIPHER_OPTION_STREAM_START	This should be passed when a new stream is starting
BLOCK_CIPHER_OPTION_STREAM_CONTINUE	The stream is still in progress.
BLOCK_CIPHER_OPTION_STREAM_COMPLETE	The stream is complete. Padding will be applied if required.
BLOCK_CIPHER_OPTION_OPTIONS_DEFAULT	A definition to specify the default set of options.
BLOCK_CIPHER_OPTION_PAD_MASK	Mask to determine the padding option that is selected.
BLOCK_CIPHER_OPTION_PAD_NONE	Pad with whatever data is already in the RAM. This flag is normally set only for the last block of data.

BLOCK_CIPHER_OPTION_PAD_NULLS	Pad with 0x00 bytes if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.
BLOCK_CIPHER_OPTION_PAD_8000	Pad with 0x80 followed by 0x00 bytes (a 1 bit followed by several 0 bits) if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.
BLOCK_CIPHER_OPTION_PAD_NUMBER	Pad with three 0x03's, four 0x04's, five 0x05's, six 0x06's, etc. set by the number of padding bytes needed if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.
BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED	The plain text pointer is pointing to data that is aligned to the target machine's word size (16-bit aligned for PIC24/dsPIC30/dsPIC33, and 8-bit aligned for PIC18). Enabling this feature may improve throughput.
BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED	The cipher text pointer is pointing to data that is aligned to the target machine's word size (16-bit aligned for PIC24/dsPIC30/dsPIC33, and 8-bit aligned for PIC18). Enabling this feature may improve throughput.
BLOCK_CIPHER_OPTION_CTR_SIZE_MASK	Mask to determine the size of the counter in bytes.
BLOCK_CIPHER_OPTION_CTR_32BIT	Treat the counter as a 32-bit counter. Leave the remaining section of the counter unchanged
BLOCK_CIPHER_OPTION_CTR_64BIT	Treat the counter as a 64-bit counter. Leave the remaining section of the counter unchanged
BLOCK_CIPHER_OPTION_CTR_128BIT	Treat the counter as a full 128-bit counter. This is the default option.

Module

General Functionality

Description

Describes general options that can be selected when encrypting/decrypting a message. Some of these options may not be necessary in certain modes of operation (for example, padding is not necessary when using OFB, which operates as a stream cipher). Note that the CTR and CFB modes have additional options that apply only to those modes.

1.7.1.1.1.1 BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY
```

Description

This option is used to pass data that will be authenticated but not encrypted into an authenticating block cipher mode function.

1.7.1.1.1.2 BLOCK_CIPHER_OPTION_STREAM_START Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_STREAM_START
```

Description

This should be passed when a new stream is starting

1.7.1.1.1.3 BLOCK_CIPHER_OPTION_STREAM_CONTINUE Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_STREAM_CONTINUE
```

Description

The stream is still in progress.

1.7.1.1.1.4 BLOCK_CIPHER_OPTION_STREAM_COMPLETE Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_STREAM_COMPLETE
```

Description

The stream is complete. Padding will be applied if required.

1.7.1.1.1.5 BLOCK_CIPHER_OPTION_OPTIONS_DEFAULT Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_OPTIONS_DEFAULT
```

Description

A definition to specify the default set of options.

1.7.1.1.1.6 BLOCK_CIPHER_OPTION_PAD_MASK Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PAD_MASK
```

Description

Mask to determine the padding option that is selected.

1.7.1.1.1.7 BLOCK_CIPHER_OPTION_PAD_NONE Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PAD_NONE
```

Description

Pad with whatever data is already in the RAM. This flag is normally set only for the last block of data.

1.7.1.1.1.8 BLOCK_CIPHER_OPTION_PAD_NULLS Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PAD_NULLS
```

Description

Pad with 0x00 bytes if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.

1.7.1.1.1.9 BLOCK_CIPHER_OPTION_PAD_8000 Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PAD_8000
```

Description

Pad with 0x80 followed by 0x00 bytes (a 1 bit followed by several 0 bits) if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.

1.7.1.1.1.10 BLOCK_CIPHER_OPTION_PAD_NUMBER Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PAD_NUMBER
```

Description

Pad with three 0x03's, four 0x04's, five 0x05's, six 0x06's, etc. set by the number of padding bytes needed if the current and previous data lengths do not end on a block boundary (multiple of 16 bytes). This flag is normally set only for the last block of data.

1.7.1.1.1.11 BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED
```

Description

The plain text pointer is pointing to data that is aligned to the target machine's word size (16-bit aligned for PIC24/dsPIC30/dsPIC33, and 8-bit aligned for PIC18). Enabling this feature may improve throughput.

1.7.1.1.1.12 BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED
```

Description

The cipher text pointer is pointing to data that is aligned to the target machine's word size (16-bit aligned for PIC24/dsPIC30/dsPIC33, and 8-bit aligned for PIC18). Enabling this feature may improve throughput.

1.7.1.1.1.13 BLOCK_CIPHER_OPTION_CTR_SIZE_MASK Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_CTR_SIZE_MASK
```

Description

Mask to determine the size of the counter in bytes.

1.7.1.1.1.14 BLOCK_CIPHER_OPTION_CTR_32BIT Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_CTR_32BIT
```

Description

Treat the counter as a 32-bit counter. Leave the remaining section of the counter unchanged

1.7.1.1.1.15 BLOCK_CIPHER_OPTION_CTR_64BIT Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_CTR_64BIT
```

Description

Treat the counter as a 64-bit counter. Leave the remaining section of the counter unchanged

1.7.1.1.1.16 BLOCK_CIPHER_OPTION_CTR_128BIT Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_OPTION_CTR_128BIT
```

Description

Treat the counter as a full 128-bit counter. This is the default option.

1.7.1.1.2 BLOCK_CIPHER_ERRORS Enumeration

File

block_cipher_16bv1.h

Syntax

```
typedef enum {
  BLOCK_CIPHER_ERROR_NONE = (0x00000000u),
  BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE,
  BLOCK_CIPHER_ERROR_CTR_COUNTER_EXPIRED,
  BLOCK_CIPHER_ERROR_INVALID_AUTHENTICATION,
  BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE,
  BLOCK_CIPHER_ERROR_INVALID_HANDLE,
  BLOCK_CIPHER_ERROR_INVALID_FUNCTION,
  BLOCK_CIPHER_ERROR_BUSY,
  BLOCK_CIPHER_ERROR_STREAM_START,
  BLOCK_CIPHER_ERROR_ABORT,
  BLOCK_CIPHER_ERROR_HW_SETTING
} BLOCK_CIPHER_ERRORS;
```

Members

Members	Description
BLOCK_CIPHER_ERROR_NONE = (0x00000000u)	No errors.
BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE	The calling function has requested that more bits be added to the key stream then are available in the buffer allotted for the key stream. Since there was not enough room to complete the request, the request was not processed.
BLOCK_CIPHER_ERROR_CTR_COUNTER_EXPIRED	The requesting call has caused the counter number to run out of unique combinations. In CTR mode it is not safe to use the same counter value for a given key.
BLOCK_CIPHER_ERROR_INVALID_AUTHENTICATION	Authentication of the specified data failed.
BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE	The specified key type (format) is unsupported by the crypto implementation that is being used.
BLOCK_CIPHER_ERROR_INVALID_HANDLE	The user specified an invalid handle
BLOCK_CIPHER_ERROR_INVALID_FUNCTION	The user specified an invalid function pointer
BLOCK_CIPHER_ERROR_BUSY	The specified block cipher module is busy and cannot start a new operation
BLOCK_CIPHER_ERROR_STREAM_START	The specified block cipher module is has already started a stream on this handle
BLOCK_CIPHER_ERROR_ABORT	The User has aborted the last operation on this handle
BLOCK_CIPHER_ERROR_HW_SETTING	The Hardware settings either key, mode, or otp program configuration is invalid

Module

General Functionality

Description

Enumeration defining potential errors the can occur when using a block cipher mode of operation. Modes that do not use keystreams will not generate errors.

1.7.1.1.3 BLOCK_CIPHER_16BV1_Close Function

Closes an opened client

File

block_cipher_16bv1.h

Syntax

```
void BLOCK_CIPHER_16BV1_Close(BLOCK_CIPHER_HANDLE handle);
```

Module

General Functionality

Returns

None

Description

Closes an opened client, resets the data structure and removes the client from the driver.

Preconditions

None.

Example

```
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

BLOCK_CIPHER_16BV1_Close (handle);
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	The handle of the opened client instance returned by BLOCK_CIPHER_16BV1_Open().

Function

void BLOCK_CIPHER_16BV1_Close (BLOCK_CIPHER_HANDLE handle)

1.7.1.1.4 BLOCK_CIPHER_16BV1_Deinitialize Function

Deinitializes the instance of the block cipher module

File

block_cipher_16bv1.h

Syntax

```
void BLOCK_CIPHER_16BV1_Deinitialize(SYS_MODULE_OBJ object);
```

Module

General Functionality

Returns

None

Description

Deinitializes the specific module instance disabling its operation.

Preconditions

None

Example

```
SYS_MODULE_OBJ sysObject;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

BLOCK_CIPHER_16BV1_Deinitialize (sysObject);
```

Parameters

Parameters	Description
SYS_MODULE_OBJ object	Identifier for the instance to be de-initialized

Function

void BLOCK_CIPHER_16BV1_Deinitialize(SYS_MODULE_OBJ object)

1.7.1.1.5 BLOCK_CIPHER_16BV1_GetState Function

Returns the current state of the client handle

File

block_cipher_16bv1.h

Syntax

```
BLOCK_CIPHER_STATE BLOCK_CIPHER_16BV1_GetState(BLOCK_CIPHER_HANDLE handle);
```

Module

General Functionality

Returns

Returns the state of the selected client handle. If the handle is in use this will be applicable to the operation being performed BLOCK_CIPHER_STATE_IDLE, BLOCK_CIPHER_STATE_ERROR, BLOCK_CIPHER_STATE_BUSY. otherwise the handle state will be BLOCK_CIPHER_STATE_CLOSED or BLOCK_CIPHER_STATE_OPEN.

Description

Returns the current state of the client handle. This can be used by the user to determine what state the thread is in durring an operation.

Preconditions

None.

Example

```
while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	The handle of the opened client instance returned by BLOCK_CIPHER_16BV1_Open().

Function

BLOCK_CIPHER_STATE BLOCK_CIPHER_16BV1_GetState (

BLOCK_CIPHER_HANDLE handle)

1.7.1.1.6 BLOCK_CIPHER_FunctionEncrypt Type

File

block_cipher_16bv1.h

Syntax

```
typedef void (* BLOCK_CIPHER_FunctionEncrypt)(BLOCK_CIPHER_HANDLE handle, void *
cipherText, void * plainText, void * key);
```

Module

General Functionality

Side Effects

None

Returns

None

Description

Function pointer for a block cipher's encryption function. When using the block cipher modes of operation module, you will configure it to use the encrypt function of the block cipher module that you are using with a pointer to that block cipher's encrypt function.

None

Remarks

None

Preconditions

None

Parameters

Parameters	Description
handle	A driver handle. If the encryption module you are using has multiple instances, this handle will be used to differentiate them. For single instance encryption modules (software-only modules) this parameter can be specified as NULL.
cipherText	The resultant cipherText produced by the encryption. The type of pointer used for this parameter will be dependent on the block cipher module you are using.
plainText	The plainText that will be encrypted. The type of pointer used for this parameter will be dependent on the block cipher module you are using.
key	Pointer to the key. The format and length of the key depends on the block cipher module you are using.

Function

void BLOCK_CIPHER_FunctionEncrypt (

BLOCK_CIPHER_HANDLE handle, void * cipherText,

void * plainText, void * key)

1.7.1.1.7 BLOCK_CIPHER_16BV1_Initialize Function

Initializes the data for the instance of the block cipher module.

File

block_cipher_16bv1.h

Syntax

```
SYS_MODULE_OBJ BLOCK_CIPHER_16BV1_Initialize(const SYS_MODULE_INDEX index, const
SYS_MODULE_INIT * const init);
```

Module

General Functionality

Returns

If successful, returns a valid handle to a driver instance object. Otherwise, it returns SYS_MODULE_OBJ_INVALID

Description

This routine initializes data for the instance of the block cipher module.

Preconditions

None

Example

```
SYS_MODULE_OBJ sysObject;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}
```

Parameters

Parameters	Description
const SYS_MODULE_INDEX index	Identifier for the instance to be initialized
const SYS_MODULE_INIT * const init	Pointer to the data structure containing any data necessary to initialize the hardware. This pointer may be null if no data is required and default initialization is to be used

Function

SYS_MODULE_OBJ BLOCK_CIPHER_16BV1_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

1.7.1.1.8 BLOCK_CIPHER_FunctionDecrypt Type

File

block_cipher_16bv1.h

Syntax

```
typedef void (* BLOCK_CIPHER_FunctionDecrypt)(BLOCK_CIPHER_HANDLE handle, void * plainText,
void * cipherText, void * key);
```

Module

General Functionality

Side Effects

None

Returns

None

Description

Function pointer for a block cipher's decryption function. When using the block cipher modes of operation module, you will configure it to use the decrypt function of the block cipher module that you are using with a pointer to that block cipher's encrypt function.

None

Remarks

None

Preconditions

None

Parameters

Parameters	Description
handle	A driver handle. If the decryption module you are using has multiple instances, this handle will be used to differentiate them. For single instance decryption modules (software-only modules) this parameter can be specified as NULL.
plainText	The resultant plainText that was decrypted. The type of pointer used for this parameter will be dependent on the block cipher module you are using.
cipherText	The cipherText that will be decrypted. The type of pointer used for this parameter will be dependent on the block cipher module you are using.
key	Pointer to the key. The format and length of the key depends on the block cipher module you are using.

Function

void BLOCK_CIPHER_FunctionDecrypt (

BLOCK_CIPHER_HANDLE handle, void * cipherText,

void * plainText, void * key)

1.7.1.1.9 BLOCK_CIPHER_16BV1_Open Function

Opens a new client for the device instance.

File

block_cipher_16bv1.h

Syntax

```
BLOCK_CIPHER_HANDLE BLOCK_CIPHER_16BV1_Open(const SYS_MODULE_INDEX index, const
DRV_IO_INTENT ioIntent);
```

Module

General Functionality

Returns

Returns a handle of the opened client instance. Otherwise, it returns BLOCK_CIPHER_HANDLE_INVALID

Description

Returns a handle of the opened client instance. All client operation APIs will require this handle as an argument.

Preconditions

The driver must have been previously initialized and in the initialized state.

Example

```
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}
```

Parameters

Parameters	Description
const SYS_MODULE_INDEX index	Identifier for the instance to opened
const DRV_IO_INTENT ioIntent	Possible values from the enumeration DRV_IO_INTENT There are currently no applicable values for this module.

Function

BLOCK_CIPHER_HANDLE BLOCK_CIPHER_16BV1_Open(const SYS_MODULE_INDEX index,

const DRV_IO_INTENT ioIntent)

1.7.1.1.10 BLOCK_CIPHER_16BV1_Tasks Function

Runs all tasks necessary for processing a request

File

block_cipher_16bv1.h

Syntax

```
void BLOCK_CIPHER_16BV1_Tasks();
```

Module

General Functionality

Returns

None

Description

This function initiates and runs all tasks necessary for processing the users request for all handles. It manages all threads and allows access to the HW to process requests. The user should call this function to keep all threads moving through the hardware. It also updates the state of each thread as requests are moved through hardware.

Preconditions

None.

Example

```
while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}
```

Function

void BLOCK_CIPHER_16BV1_Tasks (void)

1.7.1.1.11 BLOCK_CIPHER_HANDLE Type

File

block_cipher_16bv1.h

Syntax

```
typedef unsigned short int BLOCK_CIPHER_HANDLE;
```

Module

General Functionality

Description

Block cipher handle type

1.7.1.1.12 BLOCK_CIPHER_STATE Enumeration

File

block_cipher_16bv1.h

Syntax

```
typedef enum {
  BLOCK_CIPHER_STATE_CLOSED = 0,
  BLOCK_CIPHER_STATE_OPEN,
  BLOCK_CIPHER_STATE_IDLE,
  BLOCK_CIPHER_STATE_ERROR,
  BLOCK_CIPHER_STATE_BUSY
} BLOCK_CIPHER_STATE;
```

Module

General Functionality

Description

This is type BLOCK_CIPHER_STATE.

1.7.1.1.13 CRYPTO_KEY_MODE Enumeration

File

block_cipher_16bv1.h

Syntax

```
typedef enum {
  CRYPTO_MODE_NONE = 0,
  CRYPTO_64DES_1_KEY,
  CRYPTO_64DES_2_KEY,
  CRYPTO_64DES_3_KEY,
  CRYPTO_AES_128_KEY,
  CRYPTO_AES_192_KEY,
  CRYPTO_AES_256_KEY
} CRYPTO_KEY_MODE;
```

Members

Members	Description
CRYPTO_MODE_NONE = 0	Key mode is unspecified

Module

General Functionality

Description

This is type CRYPTO_KEY_MODE.

1.7.1.1.14 CRYPTO_KEY_TYPE Enumeration

File

block_cipher_16bv1.h

Syntax

```
typedef enum {
  CRYPTO_KEY_NONE = 0,
  CRYPTO_KEY_SOFTWARE,
  CRYPTO_KEY_SOFTWARE_EXPANDED,
  CRYPTO_KEY_HARDWARE_KEK,
  CRYPTO_KEY_HARDWARE_OTP_OFFSET,
  CRYPTO_KEY_HARDWARE_OTP_1,
  CRYPTO_KEY_HARDWARE_OTP_2,
  CRYPTO_KEY_HARDWARE_OTP_3,
  CRYPTO_KEY_HARDWARE_OTP_4,
  CRYPTO_KEY_HARDWARE_OTP_5,
  CRYPTO_KEY_HARDWARE_OTP_6,
  CRYPTO_KEY_HARDWARE_OTP_7
} CRYPTO_KEY_TYPE;
```

Members

Members	Description
CRYPTO_KEY_NONE = 0	Key is unspecified
CRYPTO_KEY_SOFTWARE	Key is specified in software
CRYPTO_KEY_SOFTWARE_EXPANDED	Expanded key is specified in software
CRYPTO_KEY_HARDWARE_KEK	Key-encrypted key
CRYPTO_KEY_HARDWARE_OTP_OFFSET	Key is stored in OTP memory

Module

General Functionality

Description

Enumeration defining different key types

1.7.1.1.15 _BLOCK_CIPHER_16BV1_H Macro

File

block_cipher_16bv1.h

Syntax

```
#define _BLOCK_CIPHER_16BV1_H
```

Module

General Functionality

Description

This is macro _BLOCK_CIPHER_16BV1_H.

1.7.1.1.16 BLOCK_CIPHER_HANDLE_INVALID Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_HANDLE_INVALID (-1)
```

Module

General Functionality

Description

This is macro BLOCK_CIPHER_HANDLE_INVALID.

1.7.1.1.17 BLOCK_CIPHER_INDEX Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_INDEX BLOCK_CIPHER_INDEX_0
```

Module

General Functionality

Description

Map of the default drive index to drive index 0

1.7.1.1.18 BLOCK_CIPHER_INDEX_0 Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_INDEX_0 0
```

Module

General Functionality

Description

Definition for a single drive index for the hardware AES/TDES module

1.7.1.1.19 BLOCK_CIPHER_INDEX_COUNT Macro

File

block_cipher_16bv1.h

Syntax

```
#define BLOCK_CIPHER_INDEX_COUNT 1
```

Module

General Functionality

Description

Number of drive indicies for this module

1.7.1.1.20 CRYPTO_16BV1_ALGORITHM_AES Macro

File

block_cipher_16bv1.h

Syntax

```
#define CRYPTO_16BV1_ALGORITHM_AES ((BLOCK_CIPHER_FunctionEncrypt) 0)
```

Module

General Functionality

Description

This is macro CRYPTO_16BV1_ALGORITHM_AES.

1.7.1.1.21 CRYPTO_16BV1_ALGORITHM_TDES Macro

File

block_cipher_16bv1.h

Syntax

```
#define CRYPTO_16BV1_ALGORITHM_TDES ((BLOCK_CIPHER_FunctionEncrypt) 1)
```

Module

General Functionality

Description

This is macro CRYPTO_16BV1_ALGORITHM_TDES.

1.7.1.2 ECB

Describes functionality specific to the Electronic Codebook (ECB) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_ECB_Decrypt	Decrypts cipher text using electronic codebook mode.
◆	BLOCK_CIPHER_16BV1_ECB_Encrypt	Encrypts plain text using electronic codebook mode.
◆	BLOCK_CIPHER_16BV1_ECB_Initialize	Initializes a ECB context for encryption/decryption.

Structures

Name	Description
BLOCK_CIPHER_16BV1_ECB_CONTEXT	Context structure for the electronic codebook operation

Description

Describes functionality specific to the Electronic Codebook (ECB) block cipher mode of operation.

1.7.1.2.1 BLOCK_CIPHER_16BV1_ECB_Decrypt Function

Decrypts cipher text using electronic codebook mode.

File

block_cipher_16bv1_ecb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint32_t * numPlainBytes, uint8_t * cipherText, uint32_t numCipherBytes,
uint32_t options);
```

Module

ECB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts cipher text using electronic codebook mode.

Preconditions

The ECB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// ***************************************************************
// Decrypt data in ECB mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// ECB mode context
BLOCK_CIPHER_16BV1_ECB_CONTEXT context;

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9,
0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                 0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e,
0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                 0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5,
0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                 0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad,
0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Buffer to contain encrypted plaintext
uint8_t plain_text[sizeof(cipher_text)];
// The number of bytes that were decrypted
uint32_t num_bytes_decrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_ECB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, &AESKey128, CRYPTO_KEY_SOFTWARE,
CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

// Decrypt the data
BLOCK_CIPHER_16BV1_ECB_Decrypt (handle, plain_text, &num_bytes_decrypted, cipher_text,
sizeof(cipher_text), BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * plainText	The plain test produced by the decryption. This buffer must be a multiple of the block cipher's block size, even if the cipher text passed in is not.
uint32_t * numPlainBytes	Pointer to a uint32_t; the number of bytes decrypted will be returned in this parameter.
uint8_t * cipherText	The cipher text that will be decrypted. This buffer must be a multiple of the block size, unless this is the end of the stream (the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option must be set in this case).
uint32_t numCipherBytes	The number of cipher text bytes to decrypt.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint32_t * numPlainBytes, uint8_t * cipherText,

uint32_t numCipherBytes, uint32_t options)

1.7.1.2.2 BLOCK_CIPHER_16BV1_ECB_Encrypt Function

Encrypts plain text using electronic codebook mode.

File

block_cipher_16bv1_ecb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numPlainBytes,
uint32_t options);
```

Module

ECB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts plain text using electronic codebook mode.

Preconditions

The ECB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// *****************************************************************
// Encrypt data in ECB mode with the AES algorithm.
// *****************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// ECB mode context
BLOCK_CIPHER_16BV1_ECB_CONTEXT context;

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
// The number of bytes that were encrypted
uint32_t num_bytes_encrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
```

```
and the AES block size
error = BLOCK_CIPHER_16BV1_ECB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, &AESKey128, CRYPTO_KEY_SOFTWARE,
CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Encrypt the data
BLOCK_CIPHER_16BV1_ECB_Encrypt (handle, cipher_text, &num_bytes_encrypted, (void *)
plain_text, sizeof(plain_text),
BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE|BLOCK_CIPHER_OPTION_PAD
_NONE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be a multiple of the block size, even if the plain text buffer size is not. This buffer should always be larger than the plain text buffer.
uint32_t * numCipherBytes	Pointer to a uint32_t; the number of bytes encrypted will be returned in this parameter.
uint8_t * plainText	The plain test to encrypt.
uint32_t numPlainBytes	The number of plain text bytes that must be encrypted. If the number of plain text bytes encrypted is not evenly divisible by the block size, the remaining bytes will be cached in the ECB context structure until additional data is provided.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_PAD_NONE • BLOCK_CIPHER_OPTION_PAD_NULLS • BLOCK_CIPHER_OPTION_PAD_8000 • BLOCK_CIPHER_OPTION_PAD_NUMBER • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint32_t * numCipherBytes, uint8_t * plainText,

uint32_t numPlainBytes, uint32_t options);

1.7.1.2.3 BLOCK_CIPHER_16BV1_ECB_Initialize Function

Initializes a ECB context for encryption/decryption.

File

block_cipher_16bv1_ecb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_ECB_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, void * key,
CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode);
```

Module

ECB

Returns

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a ECB context for encryption/decryption. The user will specify details about the algorithm being used in ECB mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the ECB block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_ECB_CONTEXT context;
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the block cipher module
error = BLOCK_CIPHER_16BV1_ECB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, &AESKey128, CRYPTO_KEY_SOFTWARE,
CRYPTO_AES_128_KEY);
```

```
if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_ECB_CONTEXT * context	The ECB context to initialize.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in ECB mode.
void * key	Pointer to the cryptographic key location
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_ECB_Initialize (BLOCK_CIPHER_HANDLE handle,

BLOCK_CIPHER_16BV1_ECB_CONTEXT * context,

BLOCK_CIPHER_FunctionEncrypt encryptFunction,

BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode)

1.7.1.2.4 BLOCK_CIPHER_16BV1_ECB_CONTEXT Structure

File

block_cipher_16bv1_ecb.h

Syntax

```
typedef struct {
  uint8_t remainingData[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint32_t blockSize;
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  uint8_t bytesRemaining;
  uint8_t state;
} BLOCK_CIPHER_16BV1_ECB_CONTEXT;
```

Members

Members	Description
uint8_t remainingData[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer to store data until more is avaliable if there is not enough to encrypt an entire block.
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
uint8_t bytesRemaining;	Number of bytes remaining in the remainingData buffer
uint8_t state;	The ECB mode specific state for the thread

Module

ECB

Description

Context structure for the electronic codebook operation

1.7.1.3 CBC

Describes functionality specific to the Cipher-Block Chaining (CBC) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_CBC_Decrypt	Decrypts cipher text using cipher-block chaining mode.
◆	BLOCK_CIPHER_16BV1_CBC_Encrypt	Encrypts plain text using cipher-block chaining mode.
◆	BLOCK_CIPHER_16BV1_CBC_Initialize	Initializes a CBC context for encryption/decryption.

Structures

Name	Description
BLOCK_CIPHER_16BV1_CBC_CONTEXT	Context structure for a cipher-block chaining operation

Description

Describes functionality specific to the Cipher-Block Chaining (CBC) block cipher mode of operation.

1.7.1.3.1 BLOCK_CIPHER_16BV1_CBC_Decrypt Function

Decrypts cipher text using cipher-block chaining mode.

File

block_cipher_16bv1_cbc.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint32_t * numPlainBytes, uint8_t * cipherText, uint32_t numCipherBytes,
uint32_t options);
```

Module

CBC

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts cipher text using cipher-block chaining mode.

Preconditions

The CBC context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// ***************************************************************
// Decrypt data in CBC mode with the AES algorithm.
// ***************************************************************
// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CBC mode context
BLOCK_CIPHER_16BV1_CBC_CONTEXT context;

// Initialization vector for CBC mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9,
0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                 0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e,
0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                 0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5,
0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                 0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad,
0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Buffer to contain encrypted plaintext
uint8_t plain_text[sizeof(cipher_text)];
// The number of bytes that were decrypted
uint32_t num_bytes_decrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CBC_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

// Decrypt the data
BLOCK_CIPHER_16BV1_CBC_Decrypt (handle, plain_text, &num_bytes_decrypted, cipher_text,
sizeof(cipher_text), BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * plainText	The plain test produced by the decryption. This buffer must be a multiple of the block cipher's block size, even if the cipher text passed in is not.
uint32_t * numPlainBytes	Pointer to a uint32_t; the number of bytes decrypted will be returned in this parameter.
uint8_t * cipherText	The cipher text that will be decrypted. This buffer must be a multiple of the block size, unless this is the end of the stream (the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option must be set in this case).
uint32_t numCipherBytes	The number of cipher text bytes to decrypt.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint32_t * numPlainBytes, uint8_t * cipherText,

uint32_t numCipherBytes, uint32_t options)

1.7.1.3.2 BLOCK_CIPHER_16BV1_CBC_Encrypt Function

Encrypts plain text using cipher-block chaining mode.

File

block_cipher_16bv1_cbc.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numPlainBytes,
uint32_t options);
```

Module

CBC

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts plain text using cipher-block chaining mode.

Preconditions

The CBC context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// ****************************************************************
// Encrypt data in CBC mode with the AES algorithm.
// ****************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Block cipher handle variable, to describe which AES stream to use
BLOCK_CIPHER_HANDLE handle;

// CBC mode context
BLOCK_CIPHER_16BV1_CBC_CONTEXT context;

// Initialization vector for CBC mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
// The number of bytes that were encrypted
uint32_t num_bytes_encrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CBC_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Encrypt the data
BLOCK_CIPHER_16BV1_CBC_Encrypt (handle, cipher_text, &num_bytes_encrypted, (void *)
plain_text, sizeof(plain_text),
BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE|BLOCK_CIPHER_OPTION_PAD
_NONE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be a multiple of the block size, even if the plain text buffer size is not. This buffer should always be larger than the plain text buffer.
uint32_t * numCipherBytes	Pointer to a uint32_t; the number of bytes encrypted will be returned in this parameter.
uint8_t * plainText	The plain test to encrypt.
uint32_t numPlainBytes	The number of plain text bytes that must be encrypted. If the number of plain text bytes encrypted is not evenly divisible by the block size, the remaining bytes will be cached in the CBC context structure until additional data is provided.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_PAD_NONE • BLOCK_CIPHER_OPTION_PAD_NULLS • BLOCK_CIPHER_OPTION_PAD_8000 • BLOCK_CIPHER_OPTION_PAD_NUMBER • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint32_t * numCipherBytes, uint8_t * plainText,

uint32_t numPlainBytes, uint32_t options);

1.7.1.3.3 BLOCK_CIPHER_16BV1_CBC_Initialize Function

Initializes a CBC context for encryption/decryption.

File

block_cipher_16bv1_cbc.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_CBC_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, uint8_t *
initializationVector, void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode);
```

Module

CBC

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a CBC context for encryption/decryption. The user will specify details about the algorithm being used in CBC mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the CBC block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_CBC_CONTEXT context;
// Initialization vector for CBC mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the block cipher module
error = BLOCK_CIPHER_16BV1_CBC_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_CBC_CONTEXT * context	Pointer to a context structure for this stream.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in CBC mode.
uint8_t * initializationVector	The initialization vector for this operation. The length of this vector must be equal to the block size of your block cipher.
void * key	Pointer to the cryptographic key location
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

```
    BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CBC_Initialize (BLOCK_CIPHER_HANDLE handle,

      BLOCK_CIPHER_16BV1_CBC_CONTEXT * context,

      BLOCK_CIPHER_FunctionEncrypt encryptFunction,

      BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

uint8_t * initializationVector, void * key,      CRYPTO_KEY_TYPE keyType,

      CRYPTO_KEY_MODE keyMode);
```

1.7.1.3.4 BLOCK_CIPHER_16BV1_CBC_CONTEXT Structure

File

block_cipher_16bv1_cbc.h

Syntax

```
typedef struct {
  uint8_t remainingData[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint32_t blockSize;
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  uint8_t bytesRemaining;
  uint8_t state;
} BLOCK_CIPHER_16BV1_CBC_CONTEXT;
```

Members

Members	Description
uint8_t remainingData[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer to store data until more is avaliable if there is not enough to encrypt an entire block.
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
uint8_t bytesRemaining;	Number of bytes remaining in the remainingData buffer
uint8_t state;	The CBC mode specific state for the thread

Module

CBC

Description

Context structure for a cipher-block chaining operation

1.7.1.4 CFB

Describes functionality specific to the Cipher Feedback (CFB) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_CFB_Decrypt	Decrypts cipher text using cipher-block chaining mode.
◆	BLOCK_CIPHER_16BV1_CFB_Encrypt	Encrypts plain text using cipher feedback mode.
◆	BLOCK_CIPHER_16BV1_CFB_Initialize	Initializes a CFB context for encryption/decryption.

Structures

Name	Description
BLOCK_CIPHER_16BV1_CFB_CONTEXT	Context structure for a cipher feedback operation

Description

Describes functionality specific to the Cipher Feedback (CFB) block cipher mode of operation.

1.7.1.4.1 BLOCK_CIPHER_16BV1_CFB_Decrypt Function

Decrypts cipher text using cipher-block chaining mode.

File

block_cipher_16bv1_cfb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint32_t * numPlainBytes, uint8_t * cipherText, uint32_t numCipherBytes,
uint32_t options);
```

Module

CFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts cipher text using cipher-block chaining mode.

Preconditions

The CFB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// *****************************************************************
// Decrypt data in CFB mode with the AES algorithm.
// *****************************************************************
// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CFB mode context
BLOCK_CIPHER_16BV1_CFB_CONTEXT context;

// Initialization vector for CFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9,
0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                 0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e,
0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                 0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5,
0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                 0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad,
0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Buffer to contain encrypted plaintext
uint8_t plain_text[sizeof(cipher_text)];
// The number of bytes that were decrypted
uint32_t num_bytes_decrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

// Decrypt the data
BLOCK_CIPHER_16BV1_CFB_Decrypt (handle, plain_text, &num_bytes_decrypted, cipher_text,
sizeof(cipher_text), BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use. This parameter can be specified as NULL if the block cipher does not have multiple instances.
uint8_t * plainText	The plain test produced by the decryption. This buffer must be a multiple of the block cipher's block size, even if the cipher text passed in is not.
uint32_t * numPlainBytes	Pointer to a uint32_t; the number of bytes decrypted will be returned in this parameter.
uint8_t * cipherText	The cipher text that will be decrypted. This buffer must be a multiple of the block size, unless this is the end of the stream (the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option must be set in this case).
uint32_t numCipherBytes	The number of cipher text bytes to decrypt.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_USE_CFB1 • BLOCK_CIPHER_OPTION_USE_CFB8 • BLOCK_CIPHER_OPTION_USE_CFB_BLOCK_SIZE

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint32_t * numPlainBytes, uint8_t * cipherText,

uint32_t numCipherBytes, uint32_t options)

1.7.1.4.2 BLOCK_CIPHER_16BV1_CFB_Encrypt Function

Encrypts plain text using cipher feedback mode.

File

block_cipher_16bv1_cfb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numPlainBytes,
uint32_t options);
```

Module

CFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.

- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts plain text using cipher feedback mode.

Preconditions

The CFB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

Example

```
// *****************************************************************
// Encrypt data in CFB mode with the AES algorithm.
// *****************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CFB mode context
BLOCK_CIPHER_16BV1_CFB_CONTEXT context;

// Initialization vector for CFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
// The number of bytes that were encrypted
uint32_t num_bytes_encrypted;
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Encrypt the data
BLOCK_CIPHER_16BV1_CFB_Encrypt (handle, cipher_text, &num_bytes_encrypted, (void *)
plain_text, sizeof(plain_text),
BLOCK_CIPHER_OPTION_STREAM_START|BLOCK_CIPHER_OPTION_STREAM_COMPLETE|BLOCK_CIPHER_OPTION_PAD
_NONE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use. This parameter can be specified as NULL if the block cipher does not have multiple instances.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be a multiple of the block size, even if the plain text buffer size is not. This buffer should always be larger than the plain text buffer.
uint32_t * numCipherBytes	Pointer to a uint32_t; the number of bytes encrypted will be returned in this parameter.
uint8_t * plainText	The plain test to encrypt.
uint32_t numPlainBytes	The number of plain text bytes that must be encrypted. If the number of plain text bytes encrypted is not evenly divisible by the block size, the remaining bytes will be cached in the CFB context structure until additional data is provided.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_STREAM_COMPLETE • BLOCK_CIPHER_OPTION_CIPHER_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_PLAIN_TEXT_POINTER_ALIGNED • BLOCK_CIPHER_OPTION_USE_CFB1 • BLOCK_CIPHER_OPTION_USE_CFB8 • BLOCK_CIPHER_OPTION_USE_CFB_BLOCK_SIZE

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint32_t * numCipherBytes, uint8_t * plainText,

uint32_t numPlainBytes, uint32_t options);

1.7.1.4.3 BLOCK_CIPHER_16BV1_CFB_Initialize Function

Initializes a CFB context for encryption/decryption.

File

block_cipher_16bv1_cfb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_CFB_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, uint8_t *
initializationVector, void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode);
```

Module

CFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a CFB context for encryption/decryption. The user will specify details about the algorithm being used in CFB mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the CFB block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_CFB_CONTEXT context;
// Initialization vector for CFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the block cipher module
error = BLOCK_CIPHER_16BV1_CFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, &AESKey128,
CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_CFB_CONTEXT * context	Pointer to a context structure for this stream.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in CFB mode.
uint8_t * initializationVector	The initialization vector for this operation. The length of this vector must be equal to the block size of your block cipher.
void * key	Pointer to the cryptographic key location
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CFB_Initialize (BLOCK_CIPHER_HANDLE handle,

BLOCK_CIPHER_16BV1_CFB_CONTEXT * context,

BLOCK_CIPHER_FunctionEncrypt encryptFunction,

BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

uint8_t * initialization_vector, void * key, CRYPTO_KEY_TYPE keyType,

CRYPTO_KEY_MODE keyMode);

1.7.1.4.4 BLOCK_CIPHER_16BV1_CFB_CONTEXT Structure

File

block_cipher_16bv1_cfb.h

Syntax

```
typedef struct {
  uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint32_t blockSize;
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  void * key;
  CRYPTO_KEY_TYPE keyType;
  uint8_t bytesRemaining;
  uint8_t state;
} BLOCK_CIPHER_16BV1_CFB_CONTEXT;
```

Members

Members	Description
uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Initialization vector for the CFB operation
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
void * key;	Key location

CRYPTO_KEY_TYPE keyType;	Format of the key
uint8_t bytesRemaining;	Number of bytes remaining in the remainingData buffer
uint8_t state;	The CFB mode specific state for the thread

Module

CFB

Description

Context structure for a cipher feedback operation

1.7.1.5 OFB

Describes functionality specific to the Output Feedback (OFB) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_OFB_Decrypt	Decrypts cipher text using output feedback mode.
◆	BLOCK_CIPHER_16BV1_OFB_Encrypt	Encrypts plain text using output feedback mode.
◆	BLOCK_CIPHER_16BV1_OFB_Initialize	Initializes a OFB context for encryption/decryption.
◆	BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate	Generates a key stream for use with the output feedback mode.

Macros

Name	Description
BLOCK_CIPHER_16BV1_OFB_Decrypt	This is macro BLOCK_CIPHER_16BV1_OFB_Decrypt.

Structures

Name	Description
BLOCK_CIPHER_16BV1_OFB_CONTEXT	Context structure for the output feedback operation

Description

Describes functionality specific to the Output Feedback (OFB) block cipher mode of operation.

1.7.1.5.1 BLOCK_CIPHER_16BV1_OFB_Decrypt Function

Decrypts cipher text using output feedback mode.

File

block_cipher_16bv1_ofb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint8_t * cipherText, uint32_t numBytes, uint32_t options);
```

Module

OFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts cipher text using output feedback mode.

Preconditions

The OFB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_OFB_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// ***************************************************************
// Decrypt data in OFB mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// OFB mode context
BLOCK_CIPHER_16BV1_OFB_CONTEXT context;

// Initialization vector for OFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9,
0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The decryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Buffer to contain decrypted ciphertext
uint8_t plain_text[sizeof(cipher_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_OFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
```

```
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate(handle, 4, BLOCK_CIPHER_OPTION_STREAM_START);

// Decrypt the data
BLOCK_CIPHER_16BV1_OFB_Decrypt (handle, plain_text,(void *) cipher_text,
sizeof(cipher_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * plainText	The plain text produced by the decryption. This buffer must be at least numBytes long.
uint8_t * cipherText	The cipher test to decrypt. Must be at least numBytes long.
uint32_t numBytes	The number of cipher text bytes that must be decrypted.
uint32_t options	Block cipher decryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint8_t * cipherText, uint32_t numBytes,

uint32_t options)

1.7.1.5.2 BLOCK_CIPHER_16BV1_OFB_Encrypt Function

Encrypts plain text using output feedback mode.

File

block_cipher_16bv1_ofb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numPlainBytes,
uint32_t options);
```

Module

OFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts plain text using output feedback mode.

Preconditions

The OFB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_OFB_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// ***************************************************************
// Encrypt data in OFB mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// OFB mode context
BLOCK_CIPHER_16BV1_OFB_CONTEXT context;

// Initialization vector for OFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_OFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);
```

```
if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate(handle, 4, BLOCK_CIPHER_OPTION_STREAM_START);

//Encrypt the data
BLOCK_CIPHER_16BV1_OFB_Encrypt (handle, cipher_text,(void *) plain_text,
sizeof(plain_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be at least numBytes long.
uint8_t * plainText	The plain test to encrypt. Must be at least numBytes long.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE
numBytes	The number of plain text bytes that must be encrypted.

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint8_t * plainText, uint32_t numBytes,

uint32_t options)

1.7.1.5.3 BLOCK_CIPHER_16BV1_OFB_Initialize Function

Initializes a OFB context for encryption/decryption.

File

block_cipher_16bv1_ofb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_OFB_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, uint8_t *
initializationVector, void * keyStream, uint32_t keyStreamSize, void * key, CRYPTO_KEY_TYPE
keyType, CRYPTO_KEY_MODE keyMode);
```

Module

OFB

Returns

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used

- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a OFB context for encryption/decryption. The user will specify details about the algorithm being used in OFB mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the OFB block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_OFB_CONTEXT context;
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Initialization vector for OFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Error type
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the block cipher module
error = BLOCK_CIPHER_16BV1_OFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_OFB_CONTEXT * context	The OFB context to initialize.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in OFB mode.
uint8_t * initializationVector	The initialization vector for this operation. The length of this vector must be equal to the block size of your block cipher.
void * keyStream	Pointer to a buffer to contain a calculated keyStream.
uint32_t keyStreamSize	The size of the keystream buffer, in bytes.
void * key	The cryptographic key location
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_Initialize (BLOCK_CIPHER_HANDLE handle,

BLOCK_CIPHER_16BV1_OFB_CONTEXT * context,

BLOCK_CIPHER_FunctionEncrypt encryptFunction,

BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

uint8_t * initializationVector, void * keyStream, uint32_t keyStreamSize,

void * key, CRYPTO_KEY_TYPE keyType)

1.7.1.5.4 BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate Function

Generates a key stream for use with the output feedback mode.

File

block_cipher_16bv1_ofb.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate(BLOCK_CIPHER_HANDLE handle,
uint32_t numBlocks, uint32_t options);
```

Module

OFB

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Generates a key stream for use with the output feedback mode.

Preconditions

The OFB context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_OFB_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// *****************************************************************
// Encrypt data in OFB mode with the AES algorithm.
// *****************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// OFB mode context
BLOCK_CIPHER_16BV1_OFB_CONTEXT context;

// Initialization vector for OFB mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
```

```
// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;


// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_OFB_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate(handle, 4, BLOCK_CIPHER_OPTION_STREAM_START);

//Encrypt the data
BLOCK_CIPHER_16BV1_OFB_Encrypt (handle, cipher_text,(void *) plain_text,
sizeof(plain_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint32_t numBlocks	The number of blocks of key stream that should be created. context->keyStream should have enough space remaining to handle this request.

uint32_t options	Block cipher encryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_OFB_KeyStreamGenerate (

BLOCK_CIPHER_HANDLE handle, uint32_t numBlocks, uint32_t options)

1.7.1.5.5 BLOCK_CIPHER_16BV1_OFB_CONTEXT Structure

File

block_cipher_16bv1_ofb.h

Syntax

```
typedef struct {
  uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  void * keyStream;
  void * keyStreamCurrentPosition;
  uint32_t keyStreamSize;
  uint32_t bytesRemainingInKeyStream;
  uint32_t blockSize;
  void * key;
  CRYPTO_KEY_TYPE keyType;
  uint8_t state;
} BLOCK_CIPHER_16BV1_OFB_CONTEXT;
```

Members

Members	Description
uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Initialization vector for the CFB operation
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
void * keyStream;	Pointer to the key stream. Must be a multiple of the cipher's block size, but smaller than 2^25 bytes.
void * keyStreamCurrentPosition;	Pointer to the current position in the key stream.
uint32_t keyStreamSize;	Size of the key stream.
uint32_t bytesRemainingInKeyStream;	Number of bytes remaining in the key stream
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
void * key;	Key location
CRYPTO_KEY_TYPE keyType;	Format of the key
uint8_t state;	The OFB mode specific state for the thread

Module

OFB

Description

Context structure for the output feedback operation

1.7.1.5.6 BLOCK_CIPHER_16BV1_OFB_Decrypt Macro

File

block_cipher_16bv1_ofb.h

Syntax

```
#define BLOCK_CIPHER_16BV1_OFB_Decrypt BLOCK_CIPHER_16BV1_OFB_Encrypt
```

Module

OFB

Description

This is macro BLOCK_CIPHER_16BV1_OFB_Decrypt.

1.7.1.6 CTR

Describes functionality specific to the Counter (CTR) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_CTR_Decrypt	Decrypts cipher text using counter mode.
◆	BLOCK_CIPHER_16BV1_CTR_Encrypt	Encrypts plain text using counter mode.
◆	BLOCK_CIPHER_16BV1_CTR_Initialize	Initializes a CTR context for encryption/decryption.
◆	BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate	Generates a key stream for use with the counter mode.

Macros

Name	Description
BLOCK_CIPHER_16BV1_CTR_Decrypt	This is macro BLOCK_CIPHER_16BV1_CTR_Decrypt.

Structures

Name	Description
BLOCK_CIPHER_16BV1_CTR_CONTEXT	Context structure for the counter operation

Description

Describes functionality specific to the Counter (CTR) block cipher mode of operation.

1.7.1.6.1 BLOCK_CIPHER_16BV1_CTR_Decrypt Function

Decrypts cipher text using counter mode.

File

block_cipher_16bv1_ctr.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint32_t * numPlainBytes, uint8_t * cipherText, uint32_t numBytes, uint32_t
options);
```

Module

CTR

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.

- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_CTR_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts cipher text using counter mode.

Preconditions

The CTR context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The noncePlusCounter parameter in the BLOCK_CIPHER_16BV1_CTR_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// ***************************************************************
// Decrypt data in CTR mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CTR mode context
BLOCK_CIPHER_16BV1_CTR_CONTEXT context;

// Initialization vector for CTR mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9,
0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The decryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Buffer to contain decrypted ciphertext
uint8_t plain_text[sizeof(cipher_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
```

```
and the AES block size
error = BLOCK_CIPHER_16BV1_CTR_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate(handle, 4, &round_keys,
BLOCK_CIPHER_OPTION_STREAM_START);

// Decrypt the data
BLOCK_CIPHER_16BV1_CTR_Decrypt (handle, plain_text, &numPlainBytes, (void *) cipher_text,
sizeof(cipher_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * plainText	The plain text produced by the decryption. This buffer must be at least numBytes long.
uint32_t * numPlainBytes	The number of plaintext bytes that were produced.
uint8_t * cipherText	The cipher test to decrypt. Must be at least numBytes long.
uint32_t options	Block cipher decryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_CTR_32BIT • BLOCK_CIPHER_OPTION_CTR_64BIT • BLOCK_CIPHER_OPTION_CTR_128BIT
numCipherBytes	The number of cipher text bytes that must be decrypted.

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint32_t * numPlainBytes, uint8_t * cipherText,

uint32_t numBytes, uint32_t options)

1.7.1.6.2 BLOCK_CIPHER_16BV1_CTR_Encrypt Function

Encrypts plain text using counter mode.

File

block_cipher_16bv1_ctr.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numBytes, uint32_t
options);
```

Module

CTR

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_CTR_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts plain text using counter mode.

Preconditions

The CTR context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The noncePlusCounter parameter in the BLOCK_CIPHER_16BV1_CTR_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// *****************************************************************
// Encrypt data in CTR mode with the AES algorithm.
// *****************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CTR mode context
BLOCK_CIPHER_16BV1_CTR_CONTEXT context;

// Initialization vector for CTR mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
```

```
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;
uint32_t numCipherBytes;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CTR_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate(handle, 4, &round_keys, &context,
BLOCK_CIPHER_OPTION_STREAM_START);

//Encrypt the data
BLOCK_CIPHER_16BV1_CTR_Encrypt (handle, cipher_text, &numCipherBytes, (void *) plain_text,
sizeof(plain_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
    BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be at least numBytes long.
uint32_t * numCipherBytes	The number of cipher bytes that were produced.
uint8_t * plainText	The plain test to encrypt. Must be at least numBytes long.

uint32_t options	Block cipher encryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_CTR_32BIT • BLOCK_CIPHER_OPTION_CTR_64BIT • BLOCK_CIPHER_OPTION_CTR_128BIT
numPlainBytes	The number of plain text bytes that must be encrypted.

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint32_t * numCipherBytes, uint8_t * plainText,

uint32_t numPlainBytes, uint32_t options)

1.7.1.6.3 BLOCK_CIPHER_16BV1_CTR_Initialize Function

Initializes a CTR context for encryption/decryption.

File

block_cipher_16bv1_ctr.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_CTR_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, uint8_t *
noncePlusCounter, void * keyStream, uint32_t keyStreamSize, void * key, CRYPTO_KEY_TYPE
keyType, CRYPTO_KEY_MODE keyMode);
```

Module

CTR

Returns

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a CTR context for encryption/decryption. The user will specify details about the algorithm being used in CTR mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the CTR block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_CTR_CONTEXT context;
// Initialization vector for CTR mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
```

```
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Error type
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the block cipher module
error = BLOCK_CIPHER_16BV1_CTR_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_CTR_CONTEXT * context	The CTR context to initialize.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in CTR mode.
uint8_t * noncePlusCounter	A security nonce concatenated with the initial value of the counter for this operation. The counter can be 32, 64, or 128 bits, depending on the encrypt/decrypt options selected.
void * keyStream	Pointer to a buffer to contain a calculated keyStream.
uint32_t keyStreamSize	The size of the keystream buffer, in bytes.
void * key	Pointer to the cryptographic key location
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_Initialize (BLOCK_CIPHER_HANDLE handle,

BLOCK_CIPHER_16BV1_CTR_CONTEXT * context,

BLOCK_CIPHER_FunctionEncrypt encryptFunction,

BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode)

1.7.1.6.4 BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate Function

Generates a key stream for use with the counter mode.

File

block_cipher_16bv1_ctr.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate(BLOCK_CIPHER_HANDLE handle,
uint32_t numBlocks, uint32_t options);
```

Module

CTR

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_CTR_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Generates a key stream for use with the counter mode.

Preconditions

The CTR context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The noncePlusCounter parameter in the BLOCK_CIPHER_16BV1_CTR_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// **************************************************************
// Encrypt data in CTR mode with the AES algorithm.
// **************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// CTR mode context
BLOCK_CIPHER_16BV1_CTR_CONTEXT context;

// Initialization vector for CTR mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                               0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                               0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                               0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;
uint32_t numCipherBytes;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
   // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
   // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_CTR_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, initialization_vector, (void *)&keyStream,
sizeof (keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
   // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate(handle, 4, &round_keys, &context,
BLOCK_CIPHER_OPTION_STREAM_START);

//Encrypt the data
BLOCK_CIPHER_16BV1_CTR_Encrypt (handle, cipher_text, &numCipherBytes, (void *) plain_text,
sizeof(plain_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);

while (((state = BLOCK_CIPHER_16BV1_GetState(handle)) != BLOCK_CIPHER_STATE_IDLE) && (state
!= BLOCK_CIPHER_STATE_ERROR))
{
   BLOCK_CIPHER_16BV1_Tasks();
}

if (state == BLOCK_CIPHER_STATE_ERROR)
{
   while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint32_t numBlocks	The number of blocks of key stream that should be created. context->keyStream should have enough space remaining to handle this request.

uint32_t options	Block cipher encryption options that the user can specify, or'd together. If BLOCK_CIPHER_OPTION_STREAM_START is not specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_STREAM_START • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_CTR_32BIT • BLOCK_CIPHER_OPTION_CTR_64BIT • BLOCK_CIPHER_OPTION_CTR_128BIT

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_CTR_KeyStreamGenerate (

BLOCK_CIPHER_HANDLE handle, uint32_t numBlocks, uint32_t options)

1.7.1.6.5 BLOCK_CIPHER_16BV1_CTR_CONTEXT Structure

File

block_cipher_16bv1_ctr.h

Syntax

```
typedef struct {
  uint8_t noncePlusCounter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint8_t counter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  void * keyStream;
  void * keyStreamCurrentPosition;
  uint32_t keyStreamSize;
  uint32_t bytesRemainingInKeyStream;
  uint32_t blockSize;
  void * key;
  CRYPTO_KEY_TYPE keyType;
  uint8_t state;
} BLOCK_CIPHER_16BV1_CTR_CONTEXT;
```

Members

Members	Description
uint8_t noncePlusCounter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the initial NONCE and counter.
uint8_t counter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the current counter value.
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
void * keyStream;	Pointer to the key stream. Must be a multiple of the cipher's block size, but smaller than 2^25 bytes.
void * keyStreamCurrentPosition;	Pointer to the current position in the key stream.
uint32_t keyStreamSize;	Size of the key stream.
uint32_t bytesRemainingInKeyStream;	Number of bytes remaining in the key stream
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
void * key;	Key location

CRYPTO_KEY_TYPE keyType;	Format of the key
uint8_t state;	The CTR mode specific state for the thread

Module

CTR

Description

Context structure for the counter operation

1.7.1.6.6 BLOCK_CIPHER_16BV1_CTR_Decrypt Macro

File

block_cipher_16bv1_ctr.h

Syntax

```
#define BLOCK_CIPHER_16BV1_CTR_Decrypt BLOCK_CIPHER_16BV1_CTR_Encrypt
```

Module

CTR

Description

This is macro BLOCK_CIPHER_16BV1_CTR_Decrypt.

1.7.1.7 GCM

Describes functionality specific to the Galois/Counter Mode (GCM) block cipher mode of operation.

Functions

	Name	Description
◆	BLOCK_CIPHER_16BV1_GCM_Decrypt	Decrypts/authenticates plain text using Galois/counter mode.
◆	BLOCK_CIPHER_16BV1_GCM_Encrypt	Encrypts/authenticates plain text using Galois/counter mode.
◆	BLOCK_CIPHER_16BV1_GCM_Initialize	Initializes a GCM context for encryption/decryption.
◆	BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate	Generates a key stream for use with the Galois/counter mode.

Structures

Name	Description
BLOCK_CIPHER_16BV1_GCM_CONTEXT	Context structure for the Galois counter operation

Description

Describes functionality specific to the Galois/Counter Mode (GCM) block cipher mode of operation.

1.7.1.7.1 BLOCK_CIPHER_16BV1_GCM_Decrypt Function

Decrypts/authenticates plain text using Galois/counter mode.

File

block_cipher_16bv1_gcm.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Decrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
plainText, uint32_t * numPlainBytes, uint8_t * cipherText, uint32_t numCipherBytes, uint8_t
* authenticationTag, uint8_t tagLen, uint32_t options);
```

Module

GCM

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_GCM_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_AUTHENTICATION - The calculated authentication tag did not match the one provided by the user.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Decrypts/authenticates plain text using Galois/counter mode. This function accepts a combination of data that must be authenticated but not decrypted, and data that must be authenticated and decrypted. The user should initialize a GCM context using BLOCK_CIPHER_16BV1_GCM_Initialize, then pass all authenticated-but-not-decrypted data into this function with the BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY option, and then pass any authenticated-and-decrypted data in using the BLOCK_CIPHER_OPTION_STREAM_CONTINUE option. When calling this function for the final time, the user must use the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option to generate padding required to compute the authentication tag successfully. Note that BLOCK_CIPHER_OPTION_STREAM_COMPLETE must always be specified at the end of a stream, even if no encryption is being done.

The GMAC (Galois Message Authentication Code) mode can be used by using GCM without providing any data to decrypt (e.g. by only using BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY and BLOCK_CIPHER_OPTION_STREAM_COMPLETE options).

Preconditions

The GCM context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_GCM_CONTEXT structure should be initialized. See section 8.2 of the GCM specification for more information.

Example

```
// ***************************************************************
// Decrypt data in GCM mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// GCM mode context
BLOCK_CIPHER_16BV1_GCM_CONTEXT context;

// Initialization vector for GCM mode
static uint8_t ivValue[12] = {0xca,0xfe,0xba,0xbe,0xfa,0xce,0xdb,0xad,0xde,0xca,0xf8,0x88};

// Data that will be authenticated, but not decrypted.
uint8_t authData[20] =
{0xfe,0xed,0xfa,0xce,0xde,0xad,0xbe,0xef,0xfe,0xed,0xfa,0xce,0xde,0xad,0xbe,0xef,0xab,0xad,0
xda,0xd2,};

// Cipher text to decrypt
static uint8_t cipher_text[] = { 0x42, 0x83, 0x1e, 0xc2, 0x21, 0x77, 0x74, 0x24, 0x4b,
0x72, 0x21, 0xb7, 0x84, 0xd0, 0xd4, 0x9c,
                                0xe3, 0xaa, 0x21, 0x2f, 0x2c, 0x02, 0xa4, 0xe0, 0x35, 0xc1,
0x7e, 0x23, 0x29, 0xac, 0xa1, 0x2e,
                                0x21, 0xd5, 0x14, 0xb2, 0x54, 0x66, 0x93, 0x1c, 0x7d, 0x8f,
0x6a, 0x5a, 0xac, 0x84, 0xaa, 0x05,
                                0x1b, 0xa3, 0x0b, 0x39, 0x6a, 0x0a, 0xac, 0x97, 0x3d, 0x58,
0xe0, 0x91,};


// The decryption key
static uint8_t AESKey128[] =
{0xfe,0xff,0xe9,0x92,0x86,0x65,0x73,0x1c,0x6d,0x6a,0x8f,0x94,0x67,0x30,0x83,0x08};
// Buffer to contain decrypted ciphertext
uint8_t plain_text[sizeof(cipher_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// The authentication tag for our ciphertext and our authData.
uint8_t tag[]  = {0x5b, 0xc9, 0x4f, 0xbc, 0x32, 0x21, 0xa5, 0xdb, 0x94, 0xfa, 0xe9, 0x5a,
0xe7, 0x12, 0x1a, 0x47,};
// Error type
BLOCK_CIPHER_ERRORS error;
uint32_t * numPlainBytes;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context
error = BLOCK_CIPHER_16BV1_GCM_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, (uint8_t *)ivValue, 12, (void *)&keyStream,
sizeof(keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate(handle, 4, 0);

// Authenticate the non-encrypted data
if (BLOCK_CIPHER_16BV1_GCM_Decrypt (handle, NULL, &numPlainBytes, (uint8_t *)authData, 20,
NULL, 0, BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY) != BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}

// As an example, this data will be decrypted in two blocks, to demonstrate how to use the
options.
// Decrypt the first forty bytes of data.
// Note that at this point, you don't really need to specify the tag pointer or its
length.  This parameter only
// needs to be specified when the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option is used.
if (BLOCK_CIPHER_16BV1_GCM_Decrypt (handle, plain_text, &numPlainBytes, (uint8_t
*)cipher_text, 40, tag, 16, BLOCK_CIPHER_OPTION_STREAM_CONTINUE) != BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}

// Decrypt the final twenty bytes of data.
// Since we are using BLOCK_CIPHER_OPTION_STREAM_COMPLETE, we must specify the
authentication tag and its length.  If it does not match
```

```
// the tag we obtain by decrypting the data, the Decrypt function will return
BLOCK_CIPHER_ERROR_INVALID_AUTHENTICATION.
if (BLOCK_CIPHER_16BV1_GCM_Decrypt (handle, plain_text + 40, &numPlainBytes, (uint8_t
*)cipher_text + 40, 20, tag, 16, BLOCK_CIPHER_OPTION_STREAM_COMPLETE) !=
BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * plainText	The cipher text produced by the decryption. This buffer must be at least numBytes long.
uint32_t * numPlainBytes	The number of plaintext bytes produced or the number of bytes authenticated
uint8_t * cipherText	The cipher test to decrypt. Must be at least numBytes long.
uint32_t numCipherBytes	The number of cipher text bytes that must be decrypted.
uint8_t * authenticationTag	Pointer to a structure containing the authentication tag generated by an encrypt/authenticate operation. The tag calculated during decryption will be checked against this buffer when the user specifies the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option.
uint8_t tagLen	The length of the authentication tag, in bytes.
uint32_t options	Block cipher decryption options that the user can specify, or'd together. If no option is specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_STREAM_COMPLETE

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Decrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * plainText, uint32_t * numPlainBytes, uint8_t * cipherText,

uint32_t numCipherBytes, uint8_t * authenticationTag, uint8_t tagLen,

uint32_t options)

1.7.1.7.2 BLOCK_CIPHER_16BV1_GCM_Encrypt Function

Encrypts/authenticates plain text using Galois/counter mode.

File

block_cipher_16bv1_gcm.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Encrypt(BLOCK_CIPHER_HANDLE handle, uint8_t *
cipherText, uint32_t * numCipherBytes, uint8_t * plainText, uint32_t numBytes, uint8_t *
authenticationTag, uint8_t tagLen, uint32_t options);
```

Module

GCM

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_GCM_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Encrypts/authenticates plain text using Galois/counter mode. This function accepts a combination of data that must be authenticated but not encrypted, and data that must be authenticated and encrypted. The user should initialize a GCM context using BLOCK_CIPHER_16BV1_GCM_Initialize, then pass all authenticated-but-not-encrypted data into this function with the BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY option, and then pass any authenticated-and-encrypted data in using the BLOCK_CIPHER_OPTION_STREAM_CONTINUE option. When calling this function for the final time, the user must use the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option to generate padding required to compute the authentication tag successfully. Note that BLOCK_CIPHER_OPTION_STREAM_COMPLETE must always be specified at the end of a stream, even if no encryption is being done.

The GMAC (Galois Message Authentication Code) mode can be used by using GCM without providing any data to encrypt (e.g. by only using BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY and BLOCK_CIPHER_OPTION_STREAM_COMPLETE options).

Preconditions

The GCM context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_GCM_CONTEXT structure should be initialized. See section 8.2 of the GCM specification for more information.

Example

```
// ***************************************************************
// Encrypt data in GCM mode with the AES algorithm.
// ***************************************************************

// System module object variable (for initializing AES)
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// GCM mode context
BLOCK_CIPHER_16BV1_GCM_CONTEXT context;

// Initialization vector for GCM mode
static uint8_t ivValue[12] = {0xca,0xfe,0xba,0xbe,0xfa,0xce,0xdb,0xad,0xde,0xca,0xf8,0x88};

// Data that will be authenticated, but not encrypted.
uint8_t authData[20] =
{0xfe,0xed,0xfa,0xce,0xde,0xad,0xbe,0xef,0xfe,0xed,0xfa,0xce,0xde,0xad,0xbe,0xef,0xab,0xad,0
xda,0xd2,};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
```

```
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Structure to contain the calculated authentication tag
uint8_t tag[16];
// Error type
BLOCK_CIPHER_ERRORS error;
uint32_t * numCipherBytes;

// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context
error = BLOCK_CIPHER_16BV1_GCM_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, (uint8_t *)ivValue, 12, (void *)&keyStream,
sizeof(keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate(handle, 4, &context, 0);

// Authenticate the non-encrypted data
if (BLOCK_CIPHER_16BV1_GCM_Encrypt (handle, NULL, &numCipherBytes, (uint8_t *)authData, 20,
NULL, 0, BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY) != BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}

// As an example, this data will be encrypted in two blocks, to demonstrate how to use the
options.
// Encrypt the first forty bytes of data.
// Note that at this point, you don't really need to specify the tag pointer or its
length.  This parameter only
// needs to be specified when the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option is used.
if (BLOCK_CIPHER_16BV1_GCM_Encrypt (handle, cipherText, &numCipherBytes, (uint8_t
*)ptShort, 40, tag, 16, BLOCK_CIPHER_OPTION_STREAM_CONTINUE) != BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}

//Encrypt the final twenty bytes of data.
// Since we are using BLOCK_CIPHER_OPTION_STREAM_COMPLETE, we must specify a pointer to and
length of the tag array, to store the auth tag.
if (BLOCK_CIPHER_16BV1_GCM_Encrypt (handle, cipherText + 40, &numCipherBytes, (uint8_t
*)ptShort + 40, 20, tag, 16, BLOCK_CIPHER_OPTION_STREAM_COMPLETE) !=
BLOCK_CIPHER_ERROR_NONE)
{
    // An error occured
    while(1);
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
uint8_t * cipherText	The cipher text produced by the encryption. This buffer must be at least numBytes long.
uint32_t * numCipherBytes	The number of ciphertext bytes produced or the amount of data authenticated
uint8_t * plainText	The plain test to encrypt. Must be at least numBytes long.
uint8_t * authenticationTag	Pointer to a structure to contain the authentication tag generated by a series of authentications. The tag will be written to this buffer when the user specifies the BLOCK_CIPHER_OPTION_STREAM_COMPLETE option.
uint8_t tagLen	The length of the authentication tag, in bytes. 16 bytes is standard. Shorter byte lengths can be used, but they provide less reliable authentication.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. If no option is specified then BLOCK_CIPHER_OPTION_STREAM_CONTINUE is assumed. Valid options for this function are • BLOCK_CIPHER_OPTION_AUTHENTICATE_ONLY • BLOCK_CIPHER_OPTION_STREAM_CONTINUE • BLOCK_CIPHER_OPTION_STREAM_COMPLETE
numPlainBytes	The number of plain text bytes that must be encrypted.

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Encrypt (BLOCK_CIPHER_HANDLE handle,

uint8_t * cipherText, uint32_t * numCipherBytes, uint8_t * plainText,

uint32_t numPlainBytes, uint8_t * authenticationTag, uint8_t tagLen,

uint32_t options)

1.7.1.7.3 BLOCK_CIPHER_16BV1_GCM_Initialize Function

Initializes a GCM context for encryption/decryption.

File

block_cipher_16bv1_gcm.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Initialize(BLOCK_CIPHER_HANDLE handle,
BLOCK_CIPHER_16BV1_GCM_CONTEXT * context, BLOCK_CIPHER_FunctionEncrypt encryptFunction,
BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize, uint8_t *
initializationVector, uint32_t initializationVectorLen, void * keyStream, uint32_t
keyStreamSize, void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode);
```

Module

GCM

Returns

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_UNSUPPORTED_KEY_TYPE - The specified key type is not supported by the firmware implementation being used

- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Initializes a GCM context for encryption/decryption. The user will specify details about the algorithm being used in GCM mode.

Preconditions

Any required initialization needed by the block cipher algorithm must have been performed.

Example

```
// Initialize the GCM block cipher module for use with AES.
SYS_MODULE_OBJ sysObject;
BLOCK_CIPHER_HANDLE handle;
BLOCK_CIPHER_16BV1_GCM_CONTEXT context;
// Initialization vector for GCM mode
static uint8_t ivValue[12] = {0xca,0xfe,0xba,0xbe,0xfa,0xce,0xdb,0xad,0xde,0xca,0xf8,0x88};
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
// Error type
BLOCK_CIPHER_ERRORS error;

sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context
error = BLOCK_CIPHER_16BV1_GCM_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, (uint8_t *)ivValue, 12, (void *)&keyStream,
sizeof(keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.
BLOCK_CIPHER_16BV1_GCM_CONTEXT * context	The GCM context to initialize.
BLOCK_CIPHER_FunctionEncrypt encryptFunction	Selects the algorithm that will be used
BLOCK_CIPHER_FunctionDecrypt decryptFunction	only for SW HW crypto API compatibility
uint32_t blockSize	The block size of the block cipher algorithm being used in GCM mode.
uint8_t * initializationVector	A security nonce. See the GCM specification, section 8.2 for information about constructing initialization vectors.
uint32_t initializationVectorLen	Length of the initialization vector, in bytes
void * keyStream	Pointer to a buffer to contain a calculated keyStream.
uint32_t keyStreamSize	The size of the keystream buffer, in bytes.

void * key	The key to use when encrypting/decrypting the data. The format of this key will depend on the block cipher you are using. The key is used by the Initialize function to calculate the hash subkey.
CRYPTO_KEY_TYPE keyType	The storage type of the key
CRYPTO_KEY_MODE keyMode	The length and algorithm of the key // Error type BLOCK_CIPHER_ERRORS error;

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_Initialize (BLOCK_CIPHER_HANDLE handle,

BLOCK_CIPHER_16BV1_GCM_CONTEXT * context,

BLOCK_CIPHER_FunctionEncrypt encryptFunction,

BLOCK_CIPHER_FunctionDecrypt decryptFunction, uint32_t blockSize,

uint8_t * initializationVector, void * keyStream, uint32_t keyStreamSize,

void * key, CRYPTO_KEY_TYPE keyType, CRYPTO_KEY_MODE keyMode)

1.7.1.7.4 BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate Function

Generates a key stream for use with the Galois/counter mode.

File

block_cipher_16bv1_gcm.h

Syntax

```
BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate(BLOCK_CIPHER_HANDLE handle,
uint32_t numBlocks, uint32_t options);
```

Module

GCM

Returns

Returns a member of the BLOCK_CIPHER_ERRORS enumeration:

- BLOCK_CIPHER_ERROR_NONE - no error.
- BLOCK_CIPHER_ERROR_KEY_STREAM_GEN_OUT_OF_SPACE - There was not enough room remaining in the context->keyStream buffer to fit the key data requested by the numBlocks parameter.
- BLOCK_CIPHER_ERROR_GCM_COUNTER_EXPIRED - The requesting call has caused the counter number to run out of unique combinations.
- BLOCK_CIPHER_ERROR_INVALID_HANDLE - The specified handle was invalid

Description

Generates a key stream for use with the Galois/counter mode.

Preconditions

The GCM context must be initialized with the block cipher encrypt/decrypt functions and the block cipher algorithm's block size. The block cipher module must be initialized, if necessary.

The initializationVector parameter in the BLOCK_CIPHER_16BV1_GCM_CONTEXT structure should be initialized. The size of this vector is the same as the block size of the cipher you are using.

Example

```
// *****************************************************************
// Encrypt data in GCM mode with the AES algorithm.
// *****************************************************************

// System module object variable (for initializing AES)
```

```
SYS_MODULE_OBJ sysObject;

// Drive handle variable, to describe which AES module to use
BLOCK_CIPHER_HANDLE handle;

// GCM mode context
BLOCK_CIPHER_16BV1_GCM_CONTEXT context;

// Initialization vector for GCM mode
static uint8_t initialization_vector[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

// Plain text to encrypt
static uint8_t plain_text[] = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d,
0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7,
0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb,
0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b,
0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
// The encryption key
static uint8_t AESKey128[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

// Buffer to contain encrypted plaintext
uint8_t cipher_text[sizeof(plain_text)];
//keyStream could also be allocated memory instead of fixed memory
uint8_t keyStream[AES_BLOCK_SIZE*4];
// Error type
BLOCK_CIPHER_ERRORS error;


// Initialization call for the AES module
sysObject = BLOCK_CIPHER_16BV1_Initialize (BLOCK_CIPHER_INDEX, NULL);
if (sysObject != SYS_MODULE_OBJ_STATIC)
{
    // error
}

// Driver open call for the AES module
handle = BLOCK_CIPHER_16BV1_Open (BLOCK_CIPHER_INDEX, 0);
if (handle == BLOCK_CIPHER_HANDLE_INVALID)
{
    // error
}

// Initialize the Block Cipher context with the AES module encryption/decryption functions
and the AES block size
error = BLOCK_CIPHER_16BV1_GCM_Initialize (handle, &context, CRYPTO_16BV1_ALGORITHM_AES,
CRYPTO_16BV1_ALGORITHM_AES, AES_BLOCK_SIZE, (uint8_t *)ivValue, 12, (void *)&keyStream,
sizeof(keyStream), &AESKey128, CRYPTO_KEY_SOFTWARE, CRYPTO_AES_128_KEY);

if (error != BLOCK_CIPHER_ERROR_NONE)
{
    // error
}

//Generate 4 blocks of key stream
BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate(handle, 4, BLOCK_CIPHER_OPTION_STREAM_START);

//Encrypt the data
BLOCK_CIPHER_16BV1_GCM_Encrypt (handle, cipher_text,(void *) plain_text,
sizeof(plain_text), BLOCK_CIPHER_OPTION_STREAM_CONTINUE);
```

Parameters

Parameters	Description
BLOCK_CIPHER_HANDLE handle	A handle that is passed to the block cipher's encrypt/decrypt functions to specify which instance of the block cipher module to use.

uint32_t numBlocks	The number of blocks of key stream that should be created. context->keyStream should have enough space remaining to handle this request.
uint32_t options	Block cipher encryption options that the user can specify, or'd together. This function currently does not support any options.

Function

BLOCK_CIPHER_ERRORS BLOCK_CIPHER_16BV1_GCM_KeyStreamGenerate (

BLOCK_CIPHER_HANDLE handle, uint32_t numBlocks, uint32_t options)

1.7.1.7.5 BLOCK_CIPHER_16BV1_GCM_CONTEXT Structure

File

block_cipher_16bv1_gcm.h

Syntax

```
typedef struct {
  uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint8_t counter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint8_t hashSubKey[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint8_t authTag[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  uint8_t authBuffer[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];
  BLOCK_CIPHER_FunctionEncrypt encrypt;
  BLOCK_CIPHER_FunctionDecrypt decrypt;
  void * keyStream;
  void * keyStreamCurrentPosition;
  uint32_t keyStreamSize;
  uint32_t bytesRemainingInKeyStream;
  uint32_t blockSize;
  uint32_t cipherTextLen;
  uint32_t authDataLen;
  void * key;
  CRYPTO_KEY_TYPE keyType;
  uint8_t authBufferLen;
  uint8_t * finalTagPointer;
  uint32_t finalTagLen;
  struct {
    uint8_t authCompleted : 1;
    uint8_t filler : 7;
  } flags;
  uint8_t state;
} BLOCK_CIPHER_16BV1_GCM_CONTEXT;
```

Members

Members	Description
uint8_t initializationVector[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the initialization vector and initial counter.
uint8_t counter[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the current counter value.
uint8_t hashSubKey[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the calculated hash subkey
uint8_t authTag[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing the current authentication tag
uint8_t authBuffer[CRYPTO_CONFIG_16BV1_BLOCK_MAX_SIZE];	Buffer containing data that has been encrypted but has not been authenticated
BLOCK_CIPHER_FunctionEncrypt encrypt;	Encrypt function for the algorithm being used with the block cipher mode module
BLOCK_CIPHER_FunctionDecrypt decrypt;	Decrypt function for the algorithm being used with the block cipher mode module
void * keyStream;	Pointer to the key stream. Must be a multiple of the cipher's block size, but smaller than 2^25 bytes.

void * keyStreamCurrentPosition;	Pointer to the current position in the key stream.
uint32_t keyStreamSize;	Size of the key stream.
uint32_t bytesRemainingInKeyStream;	Number of bytes remaining in the key stream
uint32_t blockSize;	Block size of the cipher algorithm being used with the block cipher mode module
uint32_t cipherTextLen;	Current number of ciphertext bytes computed
uint32_t authDataLen;	Current number of non-ciphertext bytes authenticated
void * key;	Key location
CRYPTO_KEY_TYPE keyType;	Format of the key
uint8_t authBufferLen;	Number of bytes in the auth Buffer
uint8_t authCompleted : 1;	Determines if authentication of non-encrypted data has been completed for this device.
uint8_t state;	The GCM mode specific state for the thread

Module

GCM

Description

Context structure for the Galois counter operation

Index