Arm® CryptoCell-312

Revision: r1p3

Runtime Software Developers Manual

Issue 01

101122

Non-Confidential

Arm® CryptoCell-312

Runtime Software Developers Manual

Release information

Document history

Issue	Date	Confidentiality	Change
0103-01	25-Jul-19	Non-Confidential	First release of Runtime Software Developers Manual for r1p3.

Non-Confidential Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com

Contents

1 Introduction

1.1 Product revision status

The *rmpn* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*m*	Identifies the major revision of the product, for example, r*1*.
p*n*	Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

This document is written for programmers using the CryptoCell-312 cryptographic APIs.

Familiarity with the basics of security and cryptography is assumed.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm® Glossary for more information.

1.3.2 Typographical conventions

Convention	Use
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`Monospace` **`bold`**	Denotes language keywords when used outside example code.

Convention	Use
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
⚠	Caution
✋	Warning
📝	Note

1.4 Additional reading

This document contains information that is specific to this product. See the following documents for other relevant information:

Table 1-1 Arm publications

Document name	Document ID	Licensee only Y/N
Arm® Armv8-M Architecture Reference Manual	*ARM DDI 0553A.j*	N
Arm® CryptoCell-312 Software Integrators Manual	100776	Y
Arm® Mbed™ TLS https://tls.mbed.org/	-	N

Table 1-2 Other publications

Document ID	Document name
-	*ChaCha, a variant of Salsa20*
ANSI X9.23	*Financial Institution Encryption of Wholesale Financial Messages*
Curve25519	*New Diffie-Hellman Speed Records*
FIPS 180-4	*Secure Hash Standard (SHS)*
FIPS 197	*Advanced Encryption Standard*
IEEE 1363-2000	*IEEE Standard for Standard Specifications for Public-Key Cryptography*

Document ID	Document name
IEEE 802.15.4	*IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*
IEEE P1619	*IEEE Draft Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*
ISO/IEC 18033-2:2006	*Information technology -- Security techniques -- Encryption algorithms -- Part 2: Asymmetric ciphers*
ISO/IEC 7816-4	*Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange*
NIST SP 800-108	*Recommendation for Key Derivation Using Pseudorandom Functions*
NIST SP 800-38C	*Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*
NIST SP 800-38D	*Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*
NIST SP 800-38E	*Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*
NIST SP 800-38F	*Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*
NIST SP 800-56A	*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, Rev. 2
NIST SP 800-90A	*Recommendation for Random Number Generation Using Deterministic Random Bit Generators – App C*
PKCS #1 v2.2	Public-Key Cryptography Standards *RSA Cryptography Standard*, October 2012
PKCS #3	Public-Key Cryptography Standards *Diffie Hellman Key Agreement Standard*
PKCS #7	*Cryptographic Message Syntax Standard*
RFC-2409	*The Internet Key Exchange (IKE) defines ECP group types*
RFC-2631	*Diffie-Hellman Key Agreement Method*
RFC-3394	*Advanced Encryption Standard (AES) Key Wrap Algorithm*
RFC-3526	*More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*
RFC-3610	*Counter with CBC-MAC (CCM)*
RFC-4492	*Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*
RFC-4493	*The AES-CMAC Algorithm*
RFC-5054	*Using the Secure Remote Password (SRP) Protocol for TLS Authentication*
RFC-5114	*Additional Diffie-Hellman Groups for Use with IETF Standards*
RFC-5116	*An Interface and Algorithms for Authenticated Encryption*

Document ID	Document name
RFC-5649	*Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*
RFC-5869	*HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*
RFC-6979	*Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*
RFC-7539	*ChaCha20 and Poly1305 for IETF Protocols*
RFC-7748	*Elliptic Curves for Security*
RFC-7919	*Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*
SECG SEC1	*Elliptic Curve Cryptography, 2000*

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.5.2 Feedback on content

If you have comments on content, send an e-mail to errata@arm.com and give:

- The title Arm® CryptoCell-312 Runtime Software Developers Manual.
- The number 101122.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader and cannot guarantee the quality of the represented document when used with any other PDF reader.

2 Runtime API layer

This documentation describes the runtime APIs provided by Arm CryptoCell-312. It provides the programmer with all information necessary for integrating and using the runtime APIs in the target environment.

The API layer enables using the CryptoCell cryptographic algorithms, for example, AES, hash, RSA and ECC.

Cryptographic algorithms can be divided into two main categories:

- Symmetric algorithms are mostly used for message confidentiality.

  The symmetric encryption algorithms are accessible via the generic cipher layer. For more information, see **mbedtls_cipher_setup()**.

- Asymmetric algorithms are mostly used for key exchange and message integrity.

  The asymmetric encryption algorithms are accessible via the generic public key layer.

The following algorithms are provided:

- Symmetric:
  - AES. **CryptoCell-312 hardware limitations for AES**.
- Asymmetric:
  - Diffie-Hellman-Merkle. See **mbedtls_dhm_read_public()**, **mbedtls_dhm_make_public()** and **mbedtls_dhm_calc_secret()**.
  - RSA. See **mbedtls_rsa_public()** and **mbedtls_rsa_private()**.
  - Elliptic Curves over GF(p). See **mbedtls_ecp_point_init()**.
  - Elliptic Curve Digital Signature Algorithm (ECDSA). See **mbedtls_ecdsa_init()**.
  - Elliptic Curve Diffie Hellman (ECDH). See **mbedtls_ecdh_init()**.

  The documentation is automatically generated from the source code using Doxygen.

For more information on Doxygen, see http://www.doxygen.nl.

The Module list section introduces the high-level module concepts used throughout this documentation.

2.1 Module list

Here is a list of all modules:

- CryptoCell runtime library
  - Basic CryptoCell library definitions
    - Basic CryptoCell library APIs
    - General CryptoCell definitions
    - General base error codes for CryptoCell-312
  - CryptoCell AES APIs
    - CryptoCell-312 hardware limitations for AES
    - Typical usage of AES in CryptoCell-312
    - CryptoCell AES-CCM star APIs
      - Common definitions of the CryptoCell AES-CCM
    - Definitions of CryptoCell AES APIs
      - Project definitions of CryptoCell AES APIs
  - CryptoCell DHM APIs
    - CryptoCell-312 hardware limitations for DHM
    - Typical usage of DHM in CryptoCell-312
  - CryptoCell Elliptic Curve APIs
    - ECDH module overview
      - CryptoCell-312 hardware limitations for ECDH
      - Typical usage of ECDH in CryptoCell-312
      - CryptoCell ECDH Edwards curve
    - ECDSA module overview
      - CryptoCell-312 hardware limitations for ECDSA
      - Typical usage of ECDSA in CryptoCell-312
      - CryptoCell EDDSA Edwards curve
    - CryptoCell ECIES APIs
    - CryptoCell ECPKI APIs
      - CryptoCell ECPKI supported domains
      - CryptoCell ECPKI type definitions

2.2 Data structures

The following are the data structures that are part of the delivery:

- **CC_PalTrngParams_t**
- **CCAesHwKeyData_t**
- **CCAesUserContext_t**
- **CCAesUserKeyData_t**
- **CCAssetBuff_t**
- **CCCmpuData_t**
- **CCCmpuUniqueBuff_t**
- **CCDmpuData_t**
- **CCDmpuHbkBuff_t**
- **CCEcdhFipsKatContext_t**
- **CCEcdhTempData_t**
- **CCEcdsaFipsKatContext_t**
- **CCEcdsaSignUserContext_t**
- **CCEcdsaVerifyUserContext_t**
- **CCEciesTempData_t**
- **CCEcpkiBuildTempData_t**
- **CCEcpkiDomain_t**
- **CCEcpkiKgFipsContext_t**
- **CCEcpkiKgTempData_t**
- **CCEcpkiPointAffine_t**
- **CCEcpkiPrivKey_t**
- **CCEcpkiPublKey_t**
- **CCEcpkiUserPrivKey_t**
- **CCEcpkiUserPublKey_t**
- **CCHashUserContext_t**
- **CCRndContext_t**
- **CCRndState_t**
- **CCRndWorkBuff_t**
- **CCSbCertInfo_t**

- EcdsaSignContext_t
- EcdsaVerifyContext_t
- HmacHash_t
- mbedtls_aes_context
- mbedtls_ccm_context
- mbedtls_chacha20_context
- mbedtls_chachapoly_context
- mbedtls_cipher_context_t
- mbedtls_cipher_info_t
- mbedtls_cmac_context_t
- mbedtls_ctr_drbg_context
- mbedtls_dhm_context
- mbedtls_ecdh_context
- mbedtls_ecp_curve_info
- mbedtls_ecp_group
- mbedtls_ecp_keypair
- mbedtls_ecp_point
- mbedtls_gcm_context
- mbedtls_md_context_t
- mbedtls_mng_apbc_part
- mbedtls_mng_apbcconfig
- mbedtls_nist_kw_context
- mbedtls_platform_context
- mbedtls_poly1305_context
- mbedtls_rsa_context
- mbedtls_sha1_context
- mbedtls_sha256_context
- mbedtls_sha512_context
- mbedtls_srp_context
- mbedtls_srp_group_param
- mbedtls_util_keydata

2.3 File list

The following table lists the files that are part of the delivery, and their descriptions:

Table 2-1 List of files

Filename	Description
`aes.h`	This file contains AES definitions and functions.
`bootimagesverifier_def.h`	This file contains definitions used for the Secure Boot and Secure Debug APIs.
`cc_address_defs.h`	This file contains general definitions for CryptoCell APIs.
`cc_aes_defs.h`	This file contains the type definitions that are used by the CryptoCell AES APIs.
`cc_aes_defs_proj.h`	This file contains project definitions that are used for CryptoCell AES APIs.
`cc_cmpu.h`	This file contains all of the ICV production library APIs, their enums and definitions.
`cc_dmpu.h`	This file contains all of the OEM production library APIs, their enums and definitions.
`cc_ecpki_domains_defs.h`	This file contains CryptoCell ECPKI domains supported by the project.
`cc_ecpki_types.h`	This file contains all the type definitions that are used for the CryptoCell ECPKI APIs.
`cc_error.h`	This file defines the error return code types and the numbering spaces for each module of the layers listed.
`cc_general_defs.h`	This file contains general definitions of the CryptoCell runtime SW APIs.
`cc_hash_defs.h`	This file contains definitions of the CryptoCell hash APIs.
`cc_hash_defs_proj.h`	This file contains the project-specific definitions of hash APIs.
`cc_lib.h`	This file contains all of the basic APIs of the CryptoCell library, their enums and definitions.
`cc_pal_abort.h`	This file includes all PAL abort APIs.
`cc_pal_apbc.h`	This file contains the definitions and APIs for APB-C implementation.
`cc_pal_barrier.h`	This file contains the definitions and APIs for memory-barrier implementation.
`cc_pal_compiler.h`	This file contains CryptoCell PAL platform-dependent compiler-related definitions.

Filename	Description
`cc_pal_error.h`	This file contains the error definitions of the platform-dependent PAL APIs.
`cc_pal_init.h`	This file contains the PAL layer entry point.
`cc_pal_log.h`	This file contains the PAL layer log definitions.
`cc_pal_mem.h`	This file contains functions for memory operations.
`cc_pal_memmap.h`	This file contains functions for memory mapping.
`cc_pal_mutex.h`	This file contains functions for resource management (mutex operations).
`cc_pal_pm.h`	This file contains the definitions and APIs for power-management implementation.
`cc_pal_sb_plat.h`	This file contains platform-dependent definitions used in the Boot Services code.
`cc_pal_trng.h`	This file contains APIs for retrieving TRNG user parameters.
`cc_pal_types.h`	This file contains definitions and types of CryptoCell PAL platform-dependent APIs.
`cc_pka_defs_hw.h`	This file contains all of the enums and definitions that are used in PKA APIs.
`cc_pka_hw_plat_defs.h`	This file contains the platform-dependent definitions of the CryptoCell PKA APIs.
`cc_prod.h`	This file contains all of the enums and definitions that are used for the ICV and OEM production libraries.
`cc_prod_error.h`	This file contains the error definitions of the CryptoCell production-library APIs.
`cc_rnd_common.h`	This file contains the CryptoCell random-number generation (RNG) APIs.
`cc_sram_map.h`	This file contains internal SRAM mapping definitions.
`cc_util_error.h`	This file contains the error definitions of the CryptoCell utility APIs.
`ccm.h`	This file provides an API for the CCM authenticated encryption mode for block ciphers.
`chacha20.h`	This file contains ChaCha20 definitions and functions.
`chachapoly.h`	This file contains the AEAD-ChaCha20-Poly1305 definitions and functions.
`cipher.h`	This file contains an abstraction interface for use with the cipher primitives provided by the library. It provides a common interface to all of the available cipher operations.
`cmac.h`	This file contains CMAC definitions and functions.

Filename	Description
`ctr_drbg.h`	This file contains CTR_DRBG definitions and functions.
`dhm.h`	This file contains Diffie-Hellman-Merkle (DHM) key exchange definitions and functions.
`ecdh.h`	This file contains ECDH definitions and functions.
`ecdsa.h`	This file contains ECDSA definitions and functions.
`ecp.h`	This file provides an API for Elliptic Curves over GF(P) (ECP).
`gcm.h`	This file contains GCM definitions and functions.
`hkdf.h`	This file contains the HKDF interface.
`mbedtls_aes_ext_dma.h`	This file contains all the CryptoCell AES external DMA APIs, their enums and definitions.
`mbedtls_cc_ccm_star.h`	This file contains the CryptoCell AES-CCM star APIs, their enums and definitions.
`mbedtls_cc_ecdh_edwards.h`	This file contains the CryptoCell ECDH Edwards curve APIs.
`mbedtls_cc_ecdsa_edwards.h`	This file contains the CryptoCell EDDSA Edwards curve APIs.
`mbedtls_cc_ecies.h`	This file contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.
`mbedtls_cc_mng.h`	This file contains all CryptoCell Management APIs and definitions.
`mbedtls_cc_mng_error.h`	This file contains the error definitions of the CryptoCell management APIs.
`mbedtls_cc_sbrt.h`	This file contains CryptoCell Secure Boot certificate-chain processing APIs.
`mbedtls_cc_sha512_t.h`	This file contains all of the CryptoCell SHA-512 truncated APIs, their enums and definitions.
`mbedtls_cc_srp.h`	This file contains all of the CryptoCell SRP APIs, their enums and definitions.
`mbedtls_cc_srp_error.h`	This file contains the error definitions of the CryptoCell SRP APIs.
`mbedtls_cc_util_asset_prov.h`	This file contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.
`mbedtls_cc_util_defs.h`	This file contains general definitions of the CryptoCell utility APIs.
`mbedtls_cc_util_key_derivation.h`	This file contains the CryptoCell utility key-derivation function APIs.
`mbedtls_cc_util_key_derivation_defs.h`	This file contains the definitions for the key-derivation API.

Filename	Description
`mbedtls_ccm_common.h`	This file contains the common definitions of the CryptoCell AES-CCM star APIs.
`mbedtls_chacha_ext_dma.h`	This file contains all the CryptoCell ChaCha external DMA APIs, their enums and definitions.
`mbedtls_ext_dma_error.h`	This file contains the error definitions of the CryptoCell external DMA APIs.
`mbedtls_hash_ext_dma.h`	This file contains all the CryptoCell hash external DMA APIs, their enums and definitions.
`md.h`	This file contains the generic message-digest wrapper.
`nist_kw.h`	This file provides an API for key wrapping (KW) and key wrapping with padding (KWP) as defined in *NIST SP 800-38F*. **https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf**.
`platform.h`	This file contains the definitions and functions of the Mbed TLS platform abstraction layer.
`poly1305.h`	This file contains Poly1305 definitions and functions.
`rsa.h`	This file provides an API for the RSA public-key cryptosystem.
`secureboot_defs.h`	This file contains type definitions for the Secure Boot.
`secureboot_gen_defs.h`	This file contains all of the definitions and structures used for the Secure Boot and Secure Debug.
`sha1.h`	This file contains SHA-1 definitions and functions.
`sha256.h`	This file contains SHA-224 and SHA-256 definitions and functions.
`sha512.h`	This file contains SHA-384 and SHA-512 definitions and functions.

2.4 CryptoCell runtime library

2.4.1 Modules

- **Basic CryptoCell library definitions**

  Contains basic CryptoCell library definitions.

- **CryptoCell AES APIs**

  AES is a symmetric block cipher that uses a combination of both substitution and permutation. It is fast in both software and hardware.

- **CryptoCell DHM APIs**

  Diffie-Hellman-Merkle (DHM) is used to securely exchange cryptographic keys over a public channel.

- **CryptoCell Elliptic Curve APIs**

  Contains all CryptoCell Elliptic Curve APIs.

- **CryptoCell external DMA APIs**

  Contains all CryptoCell external DMA API definitions.

- **CryptoCell hash APIs**

  Contains all CryptoCell hash APIs and definitions.

- **CryptoCell management APIs**

  Contains CryptoCell Management APIs.

- **CryptoCell PAL APIs**

  Groups all PAL APIs and definitions.

- **CryptoCell PKA APIs**

  Contains all CryptoCell PKA APIs.

- **CryptoCell RNG APIs**

  The Random Number Generator (RNG) module supports random number generation, as defined in *NIST SP 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. See **mbedtls_ctr_drbg_random()**.

- **CryptoCell RSA APIs**

  RSA is an asymmetric algorithm used for secure-data transmission.

- **CryptoCell Secure Boot and Secure Debug APIs.**

  Contains all Secure Boot and Secure Debug APIs and definitions.

- **CryptoCell SRAM mapping APIs**

  Contains internal SRAM mapping APIs.
- **CryptoCell SRP APIs**

  Contains CryptoCell SRP APIs.
- **CryptoCell utility APIs**

  This contains all utility APIs.
- **CryptoCell production-library APIs**

  Contains CryptoCell production-library APIs.

2.5 Basic CryptoCell library definitions

Contains basic CryptoCell library definitions.

2.5.1 Modules

- **Basic CryptoCell library APIs**

  Contains basic CryptoCell library APIs.
- **General CryptoCell definitions**

  Contains general definitions of the CryptoCell runtime SW APIs.
- **General base error codes for CryptoCell**

  Contains general base-error codes for CryptoCell.

2.5.2 Basic CryptoCell library APIs

Contains basic CryptoCell library APIs.

2.5.2.1 Files

- file **cc_lib.h**

  This file contains all of the basic APIs of the CryptoCell library, their enums and definitions.

2.5.2.2 Macros

- #define **DX_VERSION_PRODUCT_BIT_SHIFT** 0x18UL
- #define **DX_VERSION_PRODUCT_BIT_SIZE** 0x8UL

2.5.2.3 Enumerations

- enum **CClibRetCode_t** { **CC_LIB_RET_OK** = 0, **CC_LIB_RET_EINVAL_CTX_PTR**, **CC_LIB_RET_EINVAL_WORK_BUF_PTR**, **CC_LIB_RET_HAL**, **CC_LIB_RET_PAL**, **CC_LIB_RET_RND_INST_ERR**, **CC_LIB_RET_EINVAL_PIDR**, **CC_LIB_RET_EINVAL_CIDR**, **CC_LIB_AO_WRITE_FAILED_ERR**, **CC_LIB_RESERVE32B** = 0x7FFFFFFFL }

2.5.2.4 Functions

- **CClibRetCode_t CC_LibInit** (**CCRndContext_t** *rndContext_ptr, **CCRndWorkBuff_t** *rndWorkBuff_ptr)

  This function performs global initialization of the CryptoCell runtime library.

- **CClibRetCode_t CC_LibFini** (**CCRndContext_t** *rndContext_ptr)

  This function finalizes library operations.

2.5.2.5 Detailed description

This module lists the basic CryptoCell library APIs.

2.5.2.6 Macro definition documentation

2.5.2.6.1 #define DX_VERSION_PRODUCT_BIT_SHIFT 0x18UL

Internal definition for the product register.

2.5.2.6.2 #define DX_VERSION_PRODUCT_BIT_SIZE 0x8UL

Internal definition for the product register size.

2.5.2.7 Enumeration type documentation

2.5.2.7.1 enum CClibRetCode_t

Definitions for error returns from **CC_LibInit** or **CC_LibFini** functions.

Enumerator:

Enum	Description
`CC_LIB_RET_OK`	Success.
`CC_LIB_RET_EINVAL_CTX_PTR`	Illegal context pointer.
`CC_LIB_RET_EINVAL_WORK_BUF_PTR`	Illegal work-buffer pointer.
`CC_LIB_RET_HAL`	Error returned from the HAL layer.
`CC_LIB_RET_PAL`	Error returned from the PAL layer.

Enum	Description
`CC_LIB_RET_RND_INST_ERR`	RND instantiation failed.
`CC_LIB_RET_EINVAL_PIDR`	Invalid peripheral ID.
`CC_LIB_RET_EINVAL_CIDR`	Invalid component ID.
`CC_LIB_AO_WRITE_FAILED_ERR`	Error returned from AO write operation.
`CC_LIB_RESERVE32B`	Reserved.

2.5.2.8 Function documentation

2.5.2.8.1 CClibRetCode_t **CC_LibFini** (CCRndContext_t * **rndContext_ptr**)

It performs the following operations:

- o Frees the associated resources (mutexes).
- o Calls the HAL and PAL terminate functions.
- o Cleans the DRBG context.

Returns:

CC_LIB_RET_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer that was initialized in **CC_LibInit**.

2.5.2.8.2 CClibRetCode_t **CC_LibInit** (CCRndContext_t * **rndContext_ptr**, CCRndWorkBuff_t * **rndWorkBuff_ptr**)

It must be called once per CryptoCell cold-boot cycle. Among other initializations, this function initializes the CTR-DRBG context, including the TRNG seeding. An initialized DRBG context is required for calling DRBG APIs, as well as asymmetric-cryptography key-generation and signatures.

The primary context returned by this function can be used as a single global context for all DRBG needs. Alternatively, other contexts may be initialized and used with a more limited scope, for specific applications or specific threads.

If used, the Mutexes are initialized by this API. Therefore, unlike other APIs in the library, this API is not thread-safe.

The rndWorkBuff_ptr parameter can be NULL in case full entropy mode is used.

Returns:

CC_LIB_RET_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer allocated by the user. The context is used to maintain the RND state as well as pointers to a function used for random vector generation. This context must be saved and provided as a parameter to any API that uses the RND module.
in	`rndWorkBuff_ptr`	A scratchpad for the work of the RND module.

2.5.3 General CryptoCell definitions

Contains general definitions of the CryptoCell runtime SW APIs.

2.5.3.1 Files

- file **cc_general_defs.h**

  This file contains general definitions of the CryptoCell runtime SW APIs.
- file **cc_address_defs.h**

  This file contains general definitions for CryptoCell APIs.

2.5.3.2 Data structures

- struct **HmacHash_t**

2.5.3.3 Macros

- #define **CC_HASH_NAME_MAX_SIZE** 10
- #define **CC_AES_KDR_MAX_SIZE_BYTES** 32
- #define **CC_AES_KDR_MAX_SIZE_WORDS** (**CC_AES_KDR_MAX_SIZE_BYTES**/sizeof(uint32_t))
- #define **CC_LCS_CHIP_MANUFACTURE_LCS** 0x0
- #define **CC_LCS_SECURE_LCS** 0x5

2.5.3.4 typedefs

- typedef uint32_t **CCSramAddr_t**
- typedef uint32_t **CCDmaAddr_t**

2.5.3.5 Variables

- const **HmacHash_t HmacHashInfo_t** [**CC_HASH_NumOfModes**]
- const uint8_t **HmacSupportedHashModes_t** [**CC_HASH_NumOfModes**]
- const char **HashAlgMode2mbedtlsString** [**CC_HASH_NumOfModes**][**CC_HASH_NAME_MAX_SIZE**]

2.5.3.6 Macro definition documentation

2.5.3.6.1 #define CC_AES_KDR_MAX_SIZE_BYTES 32

Maximal size of AES HUK in bytes.

2.5.3.6.2 #define CC_AES_KDR_MAX_SIZE_WORDS (CC_AES_KDR_MAX_SIZE_BYTES/sizeof(uint32_t))

Maximal size of AES HUK in words.

2.5.3.6.3 #define CC_HASH_NAME_MAX_SIZE 10

The maximal size of the hash string.

2.5.3.6.4 #define CC_LCS_CHIP_MANUFACTURE_LCS 0x0

The Chip Manufacturer (CM) LCS value.

2.5.3.6.5 #define CC_LCS_SECURE_LCS 0x5

The Secure LCS value.

2.5.3.7 typedef documentation

2.5.3.7.1 typedef uint32_t CCDmaAddr_t

The DMA address type.

2.5.3.7.2 typedef uint32_t CCSramAddr_t

The SRAM address type.

2.5.3.8 Variable documentation

2.5.3.8.1 const char HashAlgMode2mbedtlsString[CC_HASH_NumOfModes][CC_HASH_NAME_MAX_SIZE]

Hash string names.

2.5.3.8.2 const HmacHash_t HmacHashInfo_t[CC_HASH_NumOfModes]

Hash parameters for HMAC operation.

2.5.3.8.3 const uint8_t HmacSupportedHashModes_t[CC_HASH_NumOfModes]

Supported hash modes.

2.5.4 General base error codes for CryptoCell

Contains general base-error codes for CryptoCell.

2.5.4.1 Files

- file **cc_error.h**

  This file defines the error return code types and the numbering spaces for each module of the layers listed.

2.5.4.2 Macros

- #define **CC_ERROR_BASE** 0x00F00000UL
- #define **CC_ERROR_LAYER_RANGE** 0x00010000UL
- #define **CC_ERROR_MODULE_RANGE** 0x00000100UL
- #define **CC_LAYER_ERROR_IDX** 0x00UL
- #define **LLF_LAYER_ERROR_IDX** 0x01UL
- #define **GENERIC_ERROR_IDX** 0x05UL
- #define **AES_ERROR_IDX** 0x00UL
- #define **DES_ERROR_IDX** 0x01UL
- #define **HASH_ERROR_IDX** 0x02UL
- #define **HMAC_ERROR_IDX** 0x03UL
- #define **RSA_ERROR_IDX** 0x04UL
- #define **DH_ERROR_IDX** 0x05UL
- #define **ECPKI_ERROR_IDX** 0x08UL
- #define **RND_ERROR_IDX** 0x0CUL

- #define **COMMON_ERROR_IDX** 0x0DUL
- #define **KDF_ERROR_IDX** 0x11UL
- #define **HKDF_ERROR_IDX** 0x12UL
- #define **AESCCM_ERROR_IDX** 0x15UL
- #define **FIPS_ERROR_IDX** 0x17UL
- #define **PKA_MODULE_ERROR_IDX** 0x21UL
- #define **CHACHA_ERROR_IDX** 0x22UL
- #define **EC_MONT_EDW_ERROR_IDX** 0x23UL
- #define **CHACHA_POLY_ERROR_IDX** 0x24UL
- #define **POLY_ERROR_IDX** 0x25UL
- #define **SRP_ERROR_IDX** 0x26UL
- #define **AESGCM_ERROR_IDX** 0x27UL
- #define **AES_KEYWRAP_ERROR_IDX** 0x28UL
- #define **MNG_ERROR_IDX** 0x29UL
- #define **PROD_ERROR_IDX** 0x2AUL
- #define **FFCDH_ERROR_IDX** 0x2BUL
- #define **FFC_DOMAIN_ERROR_IDX** 0x2CUL
- #define **SB_ECC_ERROR_IDX_** 0x2DUL
- #define **EXT_DMA_ERROR_IDX** 0x2EUL
- #define **CC_AES_MODULE_ERROR_BASE**
- #define **CC_DES_MODULE_ERROR_BASE**
- #define **CC_HASH_MODULE_ERROR_BASE**
- #define **CC_HMAC_MODULE_ERROR_BASE**
- #define **CC_RSA_MODULE_ERROR_BASE**
- #define **CC_DH_MODULE_ERROR_BASE**
- #define **CC_ECPKI_MODULE_ERROR_BASE**
- #define **LLF_ECPKI_MODULE_ERROR_BASE**
- #define **CC_RND_MODULE_ERROR_BASE**
- #define **LLF_RND_MODULE_ERROR_BASE**
- #define **CC_COMMON_MODULE_ERROR_BASE**
- #define **CC_KDF_MODULE_ERROR_BASE**
- #define **CC_HKDF_MODULE_ERROR_BASE**
- #define **CC_AESCCM_MODULE_ERROR_BASE**

- #define **CC_FIPS_MODULE_ERROR_BASE**
- #define **PKA_MODULE_ERROR_BASE**
- #define **CC_CHACHA_MODULE_ERROR_BASE**
- #define **CC_EC_MONT_EDW_MODULE_ERROR_BASE**
- #define **CC_CHACHA_POLY_MODULE_ERROR_BASE**
- #define **CC_POLY_MODULE_ERROR_BASE**
- #define **CC_SRP_MODULE_ERROR_BASE**
- #define **CC_AESGCM_MODULE_ERROR_BASE**
- #define **CC_AES_KEYWRAP_MODULE_ERROR_BASE**
- #define **CC_MNG_MODULE_ERROR_BASE**
- #define **CC_PROD_MODULE_ERROR_BASE**
- #define **CC_FFCDH_MODULE_ERROR_BASE**
- #define **CC_FFC_DOMAIN_MODULE_ERROR_BASE**
- #define **CC_EXT_DMA_MODULE_ERROR_BASE**
- #define **GENERIC_ERROR_BASE** (**CC_ERROR_BASE** + (**CC_ERROR_LAYER_RANGE** ***GENERIC_ERROR_IDX**))
- #define **CC_FATAL_ERROR** (**GENERIC_ERROR_BASE** + 0x00UL)
- #define **CC_OUT_OF_RESOURCE_ERROR** (**GENERIC_ERROR_BASE** + 0x01UL)
- #define **CC_ILLEGAL_RESOURCE_VAL_ERROR** (**GENERIC_ERROR_BASE** + 0x02UL)
- #define **CC_CRYPTO_RETURN_ERROR**(retCode, retcodeInfo, funcHandler) ((retCode) == 0 ? **CC_OK**: funcHandler(retCode, retcodeInfo))

2.5.4.3 Macro definition documentation

2.5.4.3.1 #define AES_ERROR_IDX 0x00UL

The AES error index.

2.5.4.3.2 #define AES_KEYWRAP_ERROR_IDX 0x28UL

The AES key-wrap error index.

2.5.4.3.3 #define AESCCM_ERROR_IDX 0x15UL

The AESCCM error index.

2.5.4.3.4 #define AESGCM_ERROR_IDX 0x27UL

The AESGCM error index.

2.5.4.3.5 #define CC_AES_KEYWRAP_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                              (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX) + \
                              (CC_ERROR_MODULE_RANGE *AES_KEYWRAP_ERROR_IDX))
```

The error base address of the AES key-wrap module - 0x00F02800.

2.5.4.3.6 #define CC_AES_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                          (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                          (CC_ERROR_MODULE_RANGE *AES_ERROR_IDX))
```

The error base address of the AES module - 0x00F00000.

2.5.4.3.7 #define CC_AESCCM_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                            (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                            (CC_ERROR_MODULE_RANGE
*AESCCM_ERROR_IDX))
```

The error base address of the AESCCM module - 0x00F01500.

2.5.4.3.8 #define CC_AESGCM_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                            (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                            (CC_ERROR_MODULE_RANGE
*AESGCM_ERROR_IDX))
```

The error base address of the AESGCM module - 0x00F02700.

2.5.4.3.9 #define CC_CHACHA_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                (CC_ERROR_MODULE_RANGE
*CHACHA_ERROR_IDX))
```

The error base address of the ChaCha module - 0x00F02200.

2.5.4.3.10 #define CC_CHACHA_POLY_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                          (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                          (CC_ERROR_MODULE_RANGE
*CHACHA_POLY_ERROR_IDX))
```

The error base address of the Chacha-POLY module - 0x00F02400.

2.5.4.3.11 #define CC_COMMON_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                  (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                                  (CC_ERROR_MODULE_RANGE
*COMMON_ERROR_IDX))
```

The error base address of the common module - 0x00F00D00.

2.5.4.3.12 #define CC_CRYPTO_RETURN_ERROR(retCode, retcodeInfo, funcHandler) ((retCode) == 0 ? CC_OK: funcHandler(retCode, retcodeInfo))

A macro that defines the CryptoCell return value.

2.5.4.3.13 #define CC_DES_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                  (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                                  (CC_ERROR_MODULE_RANGE *DES_ERROR_IDX))
```

The error base address of the DES module - 0x00F00100.

2.5.4.3.14 #define CC_DH_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX) +
\
                                                (CC_ERROR_MODULE_RANGE *DH_ERROR_IDX))
```

The error base address of the DH module - 0x00F00500.

2.5.4.3.15 #define CC_EC_MONT_EDW_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                          (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                          (CC_ERROR_MODULE_RANGE
*EC_MONT_EDW_ERROR_IDX))
```

The error base address of the EC MONT_EDW module - 0x00F02300.

2.5.4.3.16 #define CC_ECPKI_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                   (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                                   (CC_ERROR_MODULE_RANGE *ECPKI_ERROR_IDX))
```

The error base address of the ECPKI module - 0x00F00800.

2.5.4.3.17 #define CC_ERROR_BASE 0x00F00000UL

The definitions of the error number-space used for the different modules

The error base number for CryptoCell.

2.5.4.3.18 #define CC_ERROR_LAYER_RANGE 0x00010000UL

The error range number assigned for each layer.

2.5.4.3.19 #define CC_ERROR_MODULE_RANGE 0x00000100UL

The error range number assigned to each module on its specified layer.

2.5.4.3.20 #define CC_EXT_DMA_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                          (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                          (CC_ERROR_MODULE_RANGE
*EXT_DMA_ERROR_IDX))
```

The error base address of the External DMA module - 0x00F02B00.

2.5.4.3.21 #define CC_FATAL_ERROR (GENERIC_ERROR_BASE + 0x00UL)

CryptoCell fatal error.

2.5.4.3.22 #define CC_FFC_DOMAIN_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                          (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                          (CC_ERROR_MODULE_RANGE
*FFC_DOMAIN_ERROR_IDX))
```

The error base address of the FFCDH module - 0x00F02B00.

2.5.4.3.23 #define CC_FFCDH_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                             (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                             (CC_ERROR_MODULE_RANGE
*FFCDH_ERROR_IDX))
```

The error base address of the FFCDH module - 0x00F02B00.

2.5.4.3.24 #define CC_FIPS_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                         (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                         (CC_ERROR_MODULE_RANGE
*FIPS_ERROR_IDX))
```

The error base address of the FIPS module - 0x00F01700.

2.5.4.3.25 #define CC_HASH_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                       (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                       (CC_ERROR_MODULE_RANGE *HASH_ERROR_IDX))
```

The error base address of the hash module - 0x00F00200.

2.5.4.3.26 #define CC_HKDF_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX) +
\
                                    (CC_ERROR_MODULE_RANGE *HKDF_ERROR_IDX))
```

The error base address of the HKDF module - 0x00F01100.

2.5.4.3.27 #define CC_HMAC_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                       (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                       (CC_ERROR_MODULE_RANGE *HMAC_ERROR_IDX))
```

The error base address of the HMAC module - 0x00F00300.

2.5.4.3.28 #define CC_ILLEGAL_RESOURCE_VAL_ERROR (GENERIC_ERROR_BASE + 0x02UL)

CryptoCell illegal resource value error.

2.5.4.3.29 #define CC_KDF_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                        (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX) +
\
                                        (CC_ERROR_MODULE_RANGE *KDF_ERROR_IDX))
```

The error base address of the KDF module - 0x00F01100.

2.5.4.3.30 #define CC_LAYER_ERROR_IDX 0x00UL

The CryptoCell error-layer index.

2.5.4.3.31 #define CC_MNG_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                             (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX) + \
                             (CC_ERROR_MODULE_RANGE *MNG_ERROR_IDX))
```

The error base address of the Management module - 0x00F02900.

2.5.4.3.32 #define CC_OUT_OF_RESOURCE_ERROR (GENERIC_ERROR_BASE + 0x01UL)

CryptoCell out of resources error.

2.5.4.3.33 #define CC_POLY_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                    (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                    (CC_ERROR_MODULE_RANGE
*POLY_ERROR_IDX))
```

The error base address of the POLY module - 0x00F02500.

2.5.4.3.34 #define CC_PROD_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                           (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                           (CC_ERROR_MODULE_RANGE *PROD_ERROR_IDX))
```

The error base address of the production library - 0x00F02A00

2.5.4.3.35 #define CC_RND_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                           (CC_ERROR_LAYER_RANGE *CC_LAYER_ERROR_IDX)
+ \
                                        (CC_ERROR_MODULE_RANGE  *RND_ERROR_IDX))
```

The error base address of the RND module - 0x00F00C00.

2.5.4.3.36 #define CC_RSA_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                        (CC_ERROR_LAYER_RANGE  *CC_LAYER_ERROR_IDX) +
\
                                        (CC_ERROR_MODULE_RANGE  *RSA_ERROR_IDX))
```

The error base address of the RSA module - 0x00F00400.

2.5.4.3.37 #define CC_SRP_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                  (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                  (CC_ERROR_MODULE_RANGE
*SRP_ERROR_IDX))
```

The error base address of the SRP module - 0x00F02600.

2.5.4.3.38 #define CHACHA_ERROR_IDX 0x22UL

The ChaCha error index.

2.5.4.3.39 #define CHACHA_POLY_ERROR_IDX 0x24UL

The ChaCha-POLY error index.

2.5.4.3.40 #define COMMON_ERROR_IDX 0x0DUL

The Common error index.

2.5.4.3.41 #define DES_ERROR_IDX 0x01UL

The DES error index.

2.5.4.3.42 #define DH_ERROR_IDX 0x05UL

The DH error index.

2.5.4.3.43 #define EC_MONT_EDW_ERROR_IDX 0x23UL

The EC Montgomery and Edwards error index.

2.5.4.3.44 #define ECPKI_ERROR_IDX 0x08UL

The ECPKI error index.

2.5.4.3.45 #define EXT_DMA_ERROR_IDX 0x2EUL

External DMA error index.

2.5.4.3.46 #define FFC_DOMAIN_ERROR_IDX 0x2CUL

The FFC domain error index.

2.5.4.3.47 #define FFCDH_ERROR_IDX 0x2BUL

The FFCDH error index.

2.5.4.3.48 #define FIPS_ERROR_IDX 0x17UL

The FIPS error index.

2.5.4.3.49 #define GENERIC_ERROR_BASE (CC_ERROR_BASE + (CC_ERROR_LAYER_RANGE *GENERIC_ERROR_IDX))

The generic error base address of the user - 0x00F50000

2.5.4.3.50 #define GENERIC_ERROR_IDX 0x05UL

The generic error-layer index.

2.5.4.3.51 #define HASH_ERROR_IDX 0x02UL

The hash error index.

2.5.4.3.52 #define HKDF_ERROR_IDX 0x12UL

The HKDF error index.

2.5.4.3.53 #define HMAC_ERROR_IDX 0x03UL

The HMAC error index.

2.5.4.3.54 #define KDF_ERROR_IDX 0x11UL

The KDF error index.

2.5.4.3.55 #define LLF_ECPKI_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                   (CC_ERROR_LAYER_RANGE
*LLF_LAYER_ERROR_IDX) + \
                                                   (CC_ERROR_MODULE_RANGE *ECPKI_ERROR_IDX))
```

The error base address of the low-level ECPKI module - 0x00F10800.

2.5.4.3.56 #define LLF_LAYER_ERROR_IDX 0x01UL

The error-layer index for low-level functions.

2.5.4.3.57 #define LLF_RND_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                   (CC_ERROR_LAYER_RANGE
*LLF_LAYER_ERROR_IDX) + \
                                                   (CC_ERROR_MODULE_RANGE *RND_ERROR_IDX))
```

The error base address of the low-level RND module - 0x00F10C00.

2.5.4.3.58 #define MNG_ERROR_IDX 0x29UL

Management error index.

2.5.4.3.59 #define PKA_MODULE_ERROR_BASE

```
(CC_ERROR_BASE + \
                                                           (CC_ERROR_LAYER_RANGE
*CC_LAYER_ERROR_IDX) + \
                                                           (CC_ERROR_MODULE_RANGE
*PKA_MODULE_ERROR_IDX))
```

The error base address of the PKA module - 0x00F02100.

2.5.4.3.60 #define PKA_MODULE_ERROR_IDX 0x21UL

The PKA error index.

2.5.4.3.61 #define POLY_ERROR_IDX 0x25UL

The POLY error index.

2.5.4.3.62 #define PROD_ERROR_IDX 0x2AUL

Production error index.

2.5.4.3.63 #define RND_ERROR_IDX 0x0CUL

The RND error index.

2.5.4.3.64 #define RSA_ERROR_IDX 0x04UL

The RSA error index.

2.5.4.3.65 #define SB_ECC_ERROR_IDX_ 0x2DUL

Do not change. Error definition, reserved for Secure Boot ECDSA

2.5.4.3.66 #define SRP_ERROR_IDX 0x26UL

The SRP error index.

2.6 CryptoCell AES APIs

AES is a symmetric block cipher that uses a combination of both substitution and permutation. It is fast in both software and hardware.

2.6.1 Modules

- **CryptoCell-312 hardware limitations for AES**
- **Typical usage of AES in CryptoCell-312**
- **CryptoCell AES-CCM star APIs**

  Contains the CryptoCell AES-CCM star APIs.
- **Definitions of CryptoCell AES APIs**

  Contains CryptoCell AES API type definitions.

2.6.2 Detailed description

AES has a fixed block size of 128 bits, and supports the following key sizes:

- 128 bits.
- 192 bits.
- 256 bits.

  For the implementation of AES, see **aes.h**.

2.6.3 CryptoCell-312 hardware limitations for AES

The CrytoCell-312 hardware accelerates the following AES operations:

- ECB.
- CBC.
- CTR.
- CMAC. For the implementation of CMAC, see **cmac.h**.

- OFB.
- CCM. For the implementation of CCM, see **ccm.h**.
- CCM star. For the implementation of CCM star, see **mbedtls_cc_ccm_star.h** and **ccm.h**.
- GCM. For the implementation of GCM, see **gcm.h**.

To support the accelerated algorithms, the following conditions must be met:

- The input and output buffers must be DMA-able.
- The input and output buffers must be physically contingous blocks in memory.
- Buffer size must be up to 64KB.
- The context must also be DMA-able, as partial and final results are written to the context.
- Only integrated operations are supported for CCM, CCM star and GCM algorithms.

2.6.4 Typical usage of AES in CryptoCell-312

The following is a typical AES Block operation flow:

1. **mbedtls_aes_init()**.
2. **mbedtls_aes_setkey_enc()**.
3. mbedtls_aes_crypt_cbc().

2.6.5 CryptoCell AES-CCM star APIs

Contains the CryptoCell AES-CCM star APIs.

2.6.5.1 Files

- file **mbedtls_cc_ccm_star.h**

  This file contains the CryptoCell AES-CCM star APIs, their enums and definitions.

2.6.5.2 Functions

- int **mbedtls_ccm_star_nonce_generate** (unsigned char *src_addr, uint32_t frame_counter, uint8_t size_of_t, unsigned char *nonce_buf)

  This function receives the MAC source address, the frame counter, and the MAC size, and returns the required nonce for AES-CCM*, as defined in *IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).*

2.6.5.3 Function documentation

2.6.5.3.1 int mbedtls_ccm_star_nonce_generate (unsigned char * src_addr, uint32_t frame_counter, uint8_t size_of_t, unsigned char * nonce_buf)

This API should be called before **mbedtls_ccm_star_encrypt_and_tag()** or **mbedtls_ccm_star_auth_decrypt()**. The generated nonce should be provided to these functions.

Returns:

zero on success.

A non-zero value on failure, as defined in **ccm.h**.

Parameters:

Parameter	Description
`src_addr`	The MAC address in EUI-64 format.
`frame_counter`	The MAC frame counter.
`size_of_t`	The size of the AES-CCM*MAC tag in bytes: 4, 6, 8, 10, 12, 14 or 16.
`nonce_buf`	The required nonce for AES-CCM*.

2.6.5.4 Common definitions of the CryptoCell AES-CCM star APIs

Contains the CryptoCell AES-CCM star APIs.

2.6.5.4.1 Files

- file **mbedtls_ccm_common.h**

  This file contains the common definitions of the CryptoCell AES-CCM star APIs.

2.6.5.4.2 Macros

- #define **MBEDTLS_AESCCM_STAR_NONCE_SIZE_BYTES** 13
- #define **MBEDTLS_AESCCM_STAR_SOURCE_ADDRESS_SIZE_BYTES** 8
- #define **MBEDTLS_AESCCM_MODE_CCM** 0
- #define **MBEDTLS_AESCCM_MODE_STAR** 1

2.6.5.4.3 Macro definition documentation

#define MBEDTLS_AESCCM_MODE_CCM 0

AES CCM mode: CCM.

`#define MBEDTLS_AESCCM_MODE_STAR 1`

AES CCM mode: CCM star.

`#define MBEDTLS_AESCCM_STAR_NONCE_SIZE_BYTES 13`

The size of the AES CCM star nonce in bytes.

`#define MBEDTLS_AESCCM_STAR_SOURCE_ADDRESS_SIZE_BYTES 8`

The size of source address of the AES CCM star in bytes.

2.6.6 Definitions of CryptoCell AES APIs

Contains CryptoCell AES API type definitions.

2.6.6.1 Modules

- **Project definitions of CryptoCell AES APIs**

  Contains CryptoCell AES API project type definitions.

- **Common definitions of the CryptoCell AES-CCM star APIs**

  Contains the CryptoCell AES-CCM star APIs.

2.6.6.2 Files

- file **cc_aes_defs.h**

  This file contains the type definitions that are used by the CryptoCell AES APIs.

2.6.6.3 Data structures

- struct **CCAesUserContext_t**
- struct **CCAesUserKeyData_t**
- struct **CCAesHwKeyData_t**

2.6.6.4 Macros

- #define **CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS** 4
- #define **CC_AES_BLOCK_SIZE_IN_BYTES** (**CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS** *sizeof(uint32_t))
- #define **CC_AES_IV_SIZE_IN_WORDS** **CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS**
- #define **CC_AES_IV_SIZE_IN_BYTES** (**CC_AES_IV_SIZE_IN_WORDS** *sizeof(uint32_t))

2.6.6.5 typedefs

- typedef uint8_t **CCAesIv_t**[**CC_AES_IV_SIZE_IN_BYTES**]
- typedef uint8_t **CCAesKeyBuffer_t**[**CC_AES_KEY_MAX_SIZE_IN_BYTES**]

- typedef struct **CCAesUserContext_t CCAesUserContext_t**
- typedef struct **CCAesUserKeyData_t CCAesUserKeyData_t**
- typedef struct **CCAesHwKeyData_t CCAesHwKeyData_t**

2.6.6.6 Enumerations

- enum **CCAesEncryptMode_t** { **CC_AES_ENCRYPT** = 0, **CC_AES_DECRYPT** = 1, **CC_AES_NUM_OF_ENCRYPT_MODES**, **CC_AES_ENCRYPT_MODE_LAST** = 0x7FFFFFFF }
- enum **CCAesOperationMode_t** { **CC_AES_MODE_ECB** = 0, **CC_AES_MODE_CBC** = 1, **CC_AES_MODE_CBC_MAC** = 2, **CC_AES_MODE_CTR** = 3, **CC_AES_MODE_XCBC_MAC** = 4, **CC_AES_MODE_CMAC** = 5, **CC_AES_MODE_XTS** = 6, **CC_AES_MODE_CBC_CTS** = 7, **CC_AES_MODE_OFB** = 8, **CC_AES_NUM_OF_OPERATION_MODES**, **CC_AES_OPERATION_MODE_LAST** = 0x7FFFFFFF }
- enum **CCAesPaddingType_t** { **CC_AES_PADDING_NONE** = 0, **CC_AES_PADDING_PKCS7** = 1, **CC_AES_NUM_OF_PADDING_TYPES**, **CC_AES_PADDING_TYPE_LAST** = 0x7FFFFFFF }
- enum **CCAesKeyType_t** { **CC_AES_USER_KEY** = 0, **CC_AES_PLATFORM_KEY** = 1, **CC_AES_CUSTOMER_KEY** = 2, **CC_AES_NUM_OF_KEY_TYPES**, **CC_AES_KEY_TYPE_LAST** = 0x7FFFFFFF }

2.6.6.7 Macro definition documentation

2.6.6.7.1 #define CC_AES_BLOCK_SIZE_IN_BYTES (CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS *sizeof(uint32_t))

The size of the AES block in bytes.

2.6.6.7.2 #define CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS 4

The size of the AES block in words.

2.6.6.7.3 #define CC_AES_IV_SIZE_IN_BYTES (CC_AES_IV_SIZE_IN_WORDS *sizeof(uint32_t))

The size of the IV buffer in bytes.

2.6.6.7.4 #define CC_AES_IV_SIZE_IN_WORDS CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS

The size of the IV buffer in words.

2.6.6.8 typedef documentation

2.6.6.8.1 typedef struct CCAesHwKeyData_t CCAesHwKeyData_t

The AES HW key Data.

2.6.6.8.2 typedef uint8_t CCAesIv_t[CC_AES_IV_SIZE_IN_BYTES]

Defines the IV buffer. A 16-byte array.

2.6.6.8.3 typedef uint8_t CCAesKeyBuffer_t[CC_AES_KEY_MAX_SIZE_IN_BYTES]

Defines the AES key data buffer.

2.6.6.8.4 typedef struct CCAesUserContext_t CCAesUserContext_t

The context prototype of the user.

The argument type that is passed by the user to the AES APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.6.6.8.5 typedef struct CCAesUserKeyData_t CCAesUserKeyData_t

The AES key data of the user.

2.6.6.9 Enumeration type documentation

2.6.6.9.1 enum CCAesEncryptMode_t

The AES operation:

- Encrypt
- Decrypt

Enumerator:

Enum	Description
`CC_AES_ENCRYPT`	An AES encrypt operation.
`CC_AES_DECRYPT`	An AES decrypt operation.
`CC_AES_NUM_OF_ENCRYPT_MODES`	The maximal number of operations.
`CC_AES_ENCRYPT_MODE_LAST`	Reserved.

2.6.6.9.2 enum CCAesKeyType_t

The AES key type.

Enumerator:

Enum	Description
CC_AES_USER_KEY	The user key.
CC_AES_PLATFORM_KEY	The Kplt hardware key.
CC_AES_CUSTOMER_KEY	The Kcst hardware key.
CC_AES_NUM_OF_KEY_TYPES	The maximal number of AES key types.
CC_AES_KEY_TYPE_LAST	Reserved.

2.6.6.9.3 enum CCAesOperationMode_t

The AES operation mode.

Enumerator:

Enum	Description
CC_AES_MODE_ECB	ECB mode.
CC_AES_MODE_CBC	CBC mode.
CC_AES_MODE_CBC_MAC	CBC-MAC mode.
CC_AES_MODE_CTR	CTR mode.
CC_AES_MODE_XCBC_MAC	XCBC-MAC mode.
CC_AES_MODE_CMAC	CMAC mode.
CC_AES_MODE_XTS	XTS mode.
CC_AES_MODE_CBC_CTS	CBC-CTS mode.
CC_AES_MODE_OFB	OFB mode.
CC_AES_NUM_OF_OPERATION_MODES	The maximal number of AES modes.
CC_AES_OPERATION_MODE_LAST	Reserved.

2.6.6.9.4 enum CCAesPaddingType_t

The AES padding type.

Enumerator:

Enum	Description
CC_AES_PADDING_NONE	No padding.
CC_AES_PADDING_PKCS7	PKCS #7 padding.
CC_AES_NUM_OF_PADDING_TYPES	The maximal number of AES padding modes.
CC_AES_PADDING_TYPE_LAST	Reserved.

2.6.6.10 Project definitions of CryptoCell AES APIs

Contains CryptoCell AES API project type definitions.

2.6.6.10.1 Files

- file **cc_aes_defs_proj.h**

  This file contains project definitions that are used for CryptoCell AES APIs.

2.6.6.10.2 Macros

- #define **CC_AES_USER_CTX_SIZE_IN_WORDS** (4+8+8+4)
- #define **CC_AES_KEY_MAX_SIZE_IN_WORDS** 8
- #define **CC_AES_KEY_MAX_SIZE_IN_BYTES** (**CC_AES_KEY_MAX_SIZE_IN_WORDS** *sizeof(uint32_t))

2.6.6.10.3 Macro definition documentation

```
#define CC_AES_KEY_MAX_SIZE_IN_BYTES  (CC_AES_KEY_MAX_SIZE_IN_WORDS *sizeof(uint32_t))
```

The maximal size of the AES key in bytes.

```
#define CC_AES_KEY_MAX_SIZE_IN_WORDS  8
```

The maximal size of the AES key in words.

```
#define CC_AES_USER_CTX_SIZE_IN_WORDS  (4+8+8+4)
```

The size of the context prototype of the user in words. See **CCAesUserContext_t**.

2.7 CryptoCell DHM APIs

Diffie-Hellman-Merkle (DHM) is used to securely exchange cryptographic keys over a public channel.

2.7.1 Modules

- **CryptoCell-312 hardware limitations for DHM**
- **Typical usage of DHM in CryptoCell-312**

2.7.2 Detailed description

As described in *PKCS #3 Public-Key Cryptography Standards Diffie Hellman Key Agreement Standard*: "[T]wo parties, without any prior arrangements, can agree upon a secret key that is known only to them. This secret key can then be used, for example, to encrypt further communications between the parties."

The DHM module is implemented based on the definitions in the following standards:

- *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*: defines a number of standardized Diffie-Hellman groups for IKE.
- *RFC-5114 Additional Diffie-Hellman Groups for Use with IETF Standards*: defines a number of standardized Diffie-Hellman groups that can be used.

For the implementation of DHM, see **dhm.h**.

2.7.3 CryptoCell-312 hardware limitations for DHM

To support the accelerated algorithms, the following conditions must be met:

- The contexts must be DMA-able, as they might be used for some symmetric operations.

2.7.4 Typical usage of DHM in CryptoCell-312

The following is a typical DHM flow for one party:

1. **mbedtls_dhm_init()**.
2. mbedtls_mpi_read_string().
3. mbedtls_mpi_read_string().
4. **mbedtls_dhm_make_params()**.
5. **mbedtls_dhm_read_public()**.
6. **mbedtls_dhm_calc_secret()**.

2.8 CryptoCell Elliptic Curve APIs

Contains all CryptoCell Elliptic Curve APIs.

2.8.1 Modules

- **ECDH module overview**

  Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol. It allows two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

- **ECDSA module overview**

  The Elliptic Curve Digital Signature Algorithm (ECDSA) is used for generating and validating digital signatures.

- **CryptoCell ECIES APIs**

  Contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

- **CryptoCell ECPKI APIs**

  Contains all CryptoCell ECPKI APIs.

2.8.2 Detailed description

Elliptic-curve cryptography (ECC) is defined in *SECG SEC1 Elliptic Curve Cryptography*.

2.8.3 ECDH module overview

Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol. It allows two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

2.8.3.1 Modules

- **CryptoCell-312 hardware limitations for ECDH**
- **Typical usage of ECDH in CryptoCell-312**
- **CryptoCell ECDH Edwards curve APIs**

  Contains the CryptoCell ECDH Edwards curve APIs.

2.8.3.2 Detailed Description

For more information, see *NIST SP 800-56A Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography* Rev. 2.

For the implementation of ECDH, see **ecdh.h**.

2.8.3.3 CryptoCell-312 hardware limitations for ECDH

CryotoCell-312 does not support Brainpool curves.

2.8.3.4 Typical usage of ECDH in CryptoCell-312

The following is a typical ECDH operation flow:

1. **mbedtls_ecp_group_init()**.
2. mbedtls_mpi_init() for each group parameter.
3. **mbedtls_ecdh_gen_public()**.

2.8.3.5 CryptoCell ECDH Edwards curve APIs

Contains the CryptoCell ECDH Edwards curve APIs.

2.8.4 ECDSA module overview

The Elliptic Curve Digital Signature Algorithm (ECDSA) is used for generating and validating digital signatures.

2.8.4.1 Modules

- **CryptoCell-312 hardware limitations for ECDSA**
- **Typical usage of ECDSA in CryptoCell-312**
- **CryptoCell EDDSA Edwards curve APIs**

  Contains the CryptoCell EDDSA Edwards curve APIs.

2.8.4.2 Detailed Description

For the definition of ECDSA, see *SECG SEC1 Elliptic Curve Cryptography*.

For the use of ECDSA for TLS, see *RFC-4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*.

For the implementation of ECDSA, see **ecdsa.h**.

2.8.4.3 CryptoCell-312 hardware limitations for ECDSA

CryotoCell-312 does not support Brainpool curves.

Using hash functions with hash size greater than the EC modulus size is not recommended.

2.8.4.4 Typical usage of ECDSA in CryptoCell-312

The following is a typical ECDSA operation flow:

4. **mbedtls_ecp_group_init()**.
5. mbedtls_mpi_init() for each group parameter.
6. **mbedtls_ecp_gen_keypair()**.
7. **mbedtls_ecdsa_sign()** or **mbedtls_ecdsa_verify()**.

2.8.4.5 CryptoCell EDDSA Edwards curve APIs

Contains the CryptoCell EDDSA Edwards curve APIs.

2.8.4.5.1 Files

- file **mbedtls_cc_ecdsa_edwards.h**

  This file contains the CryptoCell EDDSA Edwards curve APIs.

2.8.4.5.2 Functions

- int **mbedtls_ecdsa_genkey_edwards** (**mbedtls_ecdsa_context** *ctx, **mbedtls_ecp_group_id** gid, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an EDDSA keypair on the Edwards 25519 curve.

- int **mbedtls_ecdsa_sign_edwards** (**mbedtls_ecp_group** *grp, mbedtls_mpi *r, mbedtls_mpi *s, const mbedtls_mpi *d, const unsigned char *buf, size_t blen)

  This function computes the EDDSA signature of a previously-hashed message.

- int **mbedtls_ecdsa_verify_edwards** (**mbedtls_ecp_group** *grp, const unsigned char *buf, size_t blen, const **mbedtls_ecp_point** *Q, const mbedtls_mpi *r, const mbedtls_mpi *s)

  This function verifies the EDDSA signature of a previously-hashed message.

- int **mbedtls_ecdsa_public_key_read_edwards** (**mbedtls_ecp_point** *Q, unsigned char *buf, size_t blen)

  This function imports an EC Edwards public key.

- int **mbedtls_ecdsa_public_key_write_edwards** (const **mbedtls_ecp_point** *Q, size_t *olen, unsigned char *buf, size_t blen)

  This function exports an EC Edwards public key.

2.8.4.5.3 Function Documentation

int mbedtls_ecdsa_genkey_edwards (**mbedtls_ecdsa_context** * ctx, **mbedtls_ecp_group_id** gid, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX code on failure.

Parameters:

Parameter	Description
`ctx`	The EDDSA context to store the keypair in.
`gid`	The elliptic curve to use. Currently only 25519 curve is supported.
`f_rng`	The RNG function.
`p_rng`	The RNG context.

int mbedtls_ecdsa_public_key_read_edwards (**mbedtls_ecp_point** * Q, unsigned char * buf, size_t blen)

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA or MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE on failure.

Parameters:

I/O	Parameter	Description
out	`Q`	The public key to import.
in	`buf`	The buffer to read the public key from.
in	`blen`	The length of the buffer in bytes.

int mbedtls_ecdsa_public_key_write_edwards (const **mbedtls_ecp_point** * Q, size_t * olen, unsigned char * buf, size_t blen)

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA or MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL on failure.

Parameters:

I/O	Parameter	Description
in	`Q`	The public key to export.
out	`olen`	The length of the data written in bytes.
out	`buf`	The buffer to write the public key to.
in	`blen`	The length of the buffer in bytes.

int mbedtls_ecdsa_sign_edwards (**mbedtls_ecp_group** * grp, mbedtls_mpi * r, mbedtls_mpi * s, const mbedtls_mpi * d, const unsigned char * buf, size_t blen)

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.3, step 5.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

Parameters:

Parameter	Description
`grp`	The ECP group.
`r`	The first output integer.
`s`	The second output integer.
`d`	The private signing key.
`buf`	The message hash.
`blen`	The length of buf.

int mbedtls_ecdsa_verify_edwards (**mbedtls_ecp_group** * grp, const unsigned char * buf, size_t blen, const **mbedtls_ecp_point** * Q, const mbedtls_mpi * r, const mbedtls_mpi * s)

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.4, step 3.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if signature is invalid.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure for any other reason.

Parameters:

Parameter	Description
`grp`	The ECP group.
`buf`	The message hash.
`blen`	The length of buf.
`Q`	The public key to use for verification.
`r`	The first integer of the signature.
`s`	The second integer of the signature.

2.8.5 CryptoCell ECIES APIs

Contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

2.8.5.1 Files

- file **mbedtls_cc_ecies.h**

  This file contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

2.8.5.2 Macros

- #define **MBEDTLS_ECIES_MAX_CIPHER_LEN_BYTES** ((2*__CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS__ + 1) *sizeof(int))
- #define **MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES** (sizeof(**CCEciesTempData_t**))
- #define **mbedtls_ecies_kem_encrypt**(pGrp, pRecipPublKey, kdfDerivMode, kdfHashMode, isSingleHashMode, pSecrKey, secrKeySize, pCipherData, pCipherDataSize, pBuff, buffLen, f_rng, p_rng)

  A macro for creating and encrypting a secret key.

2.8.5.3 Functions

- CCError_t **mbedtls_ecies_kem_encrypt_full** (**mbedtls_ecp_group** *pGrp, **mbedtls_ecp_point** *pRecipUzPublKey, CCKdfDerivFuncMode_t kdfDerivMode, mbedtls_hkdf_hashmode_t kdfHashMode, uint32_t isSingleHashMode, **mbedtls_ecp_point** *pExtEphUzPublicKey, mbedtls_mpi *pExtEphUzPrivateKey, uint8_t *pSecrKey, size_t secrKeySize, uint8_t *pCipherData, size_t *pCipherDataSize, void *pBuff, size_t buffLen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function creates and encrypts (encapsulates) the secret key of required size, according to *ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*, ECIES-KEM Encryption.

- CCError_t **mbedtls_ecies_kem_decrypt** (**mbedtls_ecp_group** *pGrp, mbedtls_mpi *pRecipUzPrivKey, CCKdfDerivFuncMode_t kdfDerivMode, mbedtls_hkdf_hashmode_t kdfHashMode, uint32_t isSingleHashMode, uint8_t *pCipherData, size_t cipherDataSize, uint8_t *pSecrKey, size_t secrKeySize, void *pBuff, size_t buffLen)

This function decrypts the encapsulated secret key passed by the sender, according to *ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*, sec. 10.2.4 - ECIES-KEM Decryption.

2.8.5.4 Macro Definition Documentation

2.8.5.4.1 #define mbedtls_ecies_kem_encrypt(pGrp, pRecipPublKey, kdfDerivMode, kdfHashMode, isSingleHashMode, pSecrKey, secrKeySize, pCipherData, pCipherDataSize, pBuff, buffLen, f_rng, p_rng)

```
mbedtls_ecies_kem_encrypt_full((pGrp), (pRecipPublKey), (kdfDerivMode),
(kdfHashMode), \
                    (isSingleHashMode), NULL, NULL, (pSecrKey), (secrKeySize),
\
                    (pCipherData), (pCipherDataSize), (pBuff), (buffLen), \
                    f_rng, p_rng)
```

For a description of the parameters see **mbedtls_ecies_kem_encrypt_full**.

2.8.5.4.2 #define MBEDTLS_ECIES_MAX_CIPHER_LEN_BYTES ((2*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS + 1) *sizeof(int))

The maximal length of the ECIES cipher in bytes.

2.8.5.4.3 #define MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES (sizeof(CCEciesTempData_t))

The minimal length of the ECIES buffer in bytes.

2.8.5.5 Function Documentation

2.8.5.5.1 CCError_t mbedtls_ecies_kem_decrypt (mbedtls_ecp_group * pGrp, mbedtls_mpi * pRecipUzPrivKey, CCKdfDerivFuncMode_t kdfDerivMode, mbedtls_hkdf_hashmode_t kdfHashMode, uint32_t isSingleHashMode, uint8_t * pCipherData, size_t cipherDataSize, uint8_t * pSecrKey, size_t secrKeySize, void * pBuff, size_t buffLen)

The KDF2 function mode must be used for compliance with X9.63-2011: Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography.

The term "sender" indicates an entity that creates and encapsulates the secret key using this function. The term "recipient" indicates another entity which receives and decrypts the secret key.

All public and private keys that are used must relate to the same EC Domain.

Returns:

CCError_t 0 on success.

Parameters:

I/O	Parameter	Description
in	`pGrp`	The ECP group to use.
in	`pRecipUzPrivKey`	A pointer to the private key of the recipient.
in	`kdfDerivMode`	The KDF function mode to use: KDF1 or KDF2. For more information, see CCKdfDerivFuncMode_t() in cc_kdf.h.
in	`kdfHashMode`	The used hash function.
in	`isSingleHashMode`	The specific ECIES mode definition: 0,1, according to *ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers* - section 10.2.
in	`pCipherData`	A pointer to the received encrypted cipher data.
in	`cipherDataSize`	The size of the cipher data in bytes.
in	`pSecrKey`	A pointer to the buffer for the secret-key data to be generated.
in	`secrKeySize`	The size of the secret-key data in bytes.
in	`pBuff`	A pointer to the temporary buffer.
in	`buffLen`	The size of the buffer pointed by pBuff. Must not be less than **MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES**.

2.8.5.5.2 CCError_t mbedtls_ecies_kem_encrypt_full (mbedtls_ecp_group * pGrp, mbedtls_ecp_point * pRecipUzPublKey, CCKdfDerivFuncMode_t kdfDerivMode, mbedtls_hkdf_hashmode_t kdfHashMode, uint32_t isSingleHashMode, mbedtls_ecp_point * pExtEphUzPublicKey, mbedtls_mpi * pExtEphUzPrivateKey, uint8_t * pSecrKey, size_t secrKeySize, uint8_t * pCipherData, size_t * pCipherDataSize, void * pBuff, size_t buffLen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

To call this function in applications, the **mbedtls_ecies_kem_encrypt** macro definition must be used. The function itself has the additional input of the external ephemeral key pair, used only for testing purposes.

Use KDF2 function mode for compliance with X9.63-2011: Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography.

The term "sender" indicates an entity that creates and encapsulates the secret key using this function. The term "recipient" indicates another entity which receives and decrypts the secret key.

All public and private keys that are used must relate to the same EC Domain.

The user must verify that the public key of the recipient is on the elliptic curve before it is used in this function.

Returns:

CCError_t 0 on success.

Parameters:

I/O	Parameter	Description
in	`pGrp`	The ECP group to use.
in	`pRecipUzPublKey`	A pointer to the public key of the recipient.
in	`kdfDerivMode`	The KDF function mode to use: KDF1 or KDF2. For more information, see CCKdfDerivFuncMode_t() in cc_kdf.h.
in	`kdfHashMode`	The used hash function.
in	`isSingleHashMode`	The specific ECIES mode, according to *ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers* - section 10.2: 0: Not-single hash, or 1: Single hash.
in	`pExtEphUzPublicKey`	A pointer to the ephemeral public key related to the private key. Must be set to NULL if pExtEphUzPrivateKey = NULL.
in	`pExtEphUzPrivateKey`	The pointer to the external ephemeral private key. This key is used only for testing the function. In regular use, the pointer should be set to NULL and then the random key-pair should be generated internally.
in	`pSecrKey`	A pointer to the buffer for the secret-key data to be generated.
in	`secrKeySize`	The size of the secret-key data in bytes.
in	`pCipherData`	A pointer to the encrypted cipher text.
in,out	`pCipherDataSize`	In: A pointer to the size of the buffer for CipherData output, or Out: The size of the buffer for CipherData output in bytes.
in	`pBuff`	A pointer to the temporary buffer.
in	`buffLen`	The size of the buffer pointed by pBuff. Must not be less than **MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES**.
in	`f_rng`	The RNG function required for generating a key pair when pExtEphUzPublicKey and pExtEphUzPrivateKey are NULL
in	`p_rng`	The RNG parameter.

2.8.6 CryptoCell ECPKI APIs

Contains all CryptoCell ECPKI APIs.

2.8.6.1 Modules

- **CryptoCell ECPKI supported domains**

  Contains CryptoCell ECPKI domains supported by the project.

- **CryptoCell ECPKI type definitions**

Contains CryptoCell ECPKI API type definitions.

2.8.6.2 Detailed Description

This module contains all definitions relating to Elliptic Curve Public Key Infrastructure.

2.8.6.3 CryptoCell ECPKI supported domains

Contains CryptoCell ECPKI domains supported by the project.

2.8.6.3.1 Files

- file **cc_ecpki_domains_defs.h**

  This file contains CryptoCell ECPKI domains supported by the project.

2.8.6.3.2 Typedefs

- typedef const **CCEcpkiDomain_t** *(***getDomainFuncP**) (void)

2.8.6.3.3 Typedef Documentation

typedef const **CCEcpkiDomain_t***(*getDomainFuncP) (void)

Definition of the domain-retrieval function.

2.8.6.4 CryptoCell ECPKI type definitions

Contains CryptoCell ECPKI API type definitions.

2.8.6.4.1 Files

- file **cc_ecpki_types.h**

  This file contains all the type definitions that are used for the CryptoCell ECPKI APIs.

2.8.6.4.2 Data Structures

- struct **CCEcpkiDomain_t**

  The structure containing the EC domain parameters in little-endian form.

- struct **CCEcpkiPointAffine_t**
- struct **CCEcpkiPublKey_t**
- struct **CCEcpkiUserPublKey_t**

  The user structure prototype of the EC public key.

- struct **CCEcpkiPrivKey_t**
- struct **CCEcpkiUserPrivKey_t**

The user structure prototype of the EC private key.

- struct **CCEcdhTempData_t**
- struct **CCEcpkiBuildTempData_t**
- struct **EcdsaSignContext_t**
- struct **CCEcdsaSignUserContext_t**

  The context definition of the user for the signing operation.
- struct **EcdsaVerifyContext_t**
- struct **CCEcdsaVerifyUserContext_t**

  The context definition of the user for the verification operation.
- struct **CCEcpkiKgTempData_t**
- struct **CCEciesTempData_t**
- struct **CCEcpkiKgFipsContext_t**
- struct **CCEcdsaFipsKatContext_t**
- struct **CCEcdhFipsKatContext_t**

2.8.6.4.3 Macros

- #define **CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS** (10 + 3***CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**)
- #define **CC_ECPKI_FIPS_ORDER_LENGTH** (256/**CC_BITS_IN_BYTE**)

2.8.6.4.4 Typedefs

- typedef struct **CCEcpkiUserPublKey_t CCEcpkiUserPublKey_t**

  The user structure prototype of the EC public key.
- typedef struct **CCEcpkiUserPrivKey_t CCEcpkiUserPrivKey_t**

  The user structure prototype of the EC private key.
- typedef struct **CCEcdhTempData_t CCEcdhTempData_t**
- typedef struct **CCEcpkiBuildTempData_t CCEcpkiBuildTempData_t**
- typedef uint32_t **CCEcdsaSignIntBuff_t**[**CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS**]
- typedef struct **CCEcdsaSignUserContext_t CCEcdsaSignUserContext_t**

  The context definition of the user for the signing operation.
- typedef uint32_t **CCEcdsaVerifyIntBuff_t**[**CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS**]
- typedef struct **CCEcdsaVerifyUserContext_t CCEcdsaVerifyUserContext_t**

The context definition of the user for the verification operation.

- typedef struct **CCEcpkiKgTempData_t CCEcpkiKgTempData_t**
- typedef struct **CCEciesTempData_t CCEciesTempData_t**
- typedef struct **CCEcpkiKgFipsContext_t CCEcpkiKgFipsContext_t**
- typedef struct **CCEcdsaFipsKatContext_t CCEcdsaFipsKatContext_t**
- typedef struct **CCEcdhFipsKatContext_t CCEcdhFipsKatContext_t**

2.8.6.4.5 Enumerations

- enum **CCEcpkiDomainID_t** { **CC_ECPKI_DomainID_secp192k1**, **CC_ECPKI_DomainID_secp192r1**, **CC_ECPKI_DomainID_secp224k1**, **CC_ECPKI_DomainID_secp224r1**, **CC_ECPKI_DomainID_secp256k1**, **CC_ECPKI_DomainID_secp256r1**, **CC_ECPKI_DomainID_secp384r1**, **CC_ECPKI_DomainID_secp521r1**, **CC_ECPKI_DomainID_OffMode**, **CC_ECPKI_DomainIDLast** = 0x7FFFFFFF }

  EC domain idetifiers.

- enum **CCEcpkiHashOpMode_t** { **CC_ECPKI_HASH_SHA1_mode** = 0, **CC_ECPKI_HASH_SHA224_mode** = 1, **CC_ECPKI_HASH_SHA256_mode** = 2, **CC_ECPKI_HASH_SHA384_mode** = 3, **CC_ECPKI_HASH_SHA512_mode** = 4, **CC_ECPKI_AFTER_HASH_SHA1_mode** = 5, **CC_ECPKI_AFTER_HASH_SHA224_mode** = 6, **CC_ECPKI_AFTER_HASH_SHA256_mode** = 7, **CC_ECPKI_AFTER_HASH_SHA384_mode** = 8, **CC_ECPKI_AFTER_HASH_SHA512_mode** = 9, **CC_ECPKI_HASH_NumOfModes**, **CC_ECPKI_HASH_OpModeLast** = 0x7FFFFFFF }

  Hash operation mode.

- enum **CCEcpkiPointCompression_t** { **CC_EC_PointCompressed** = 2, **CC_EC_PointUncompressed** = 4, **CC_EC_PointContWrong** = 5, **CC_EC_PointHybrid** = 6, **CC_EC_PointCompresOffMode** = 8, **CC_ECPKI_PointCompressionLast** = 0x7FFFFFFF }
- enum **ECPublKeyCheckMode_t** { **CheckPointersAndSizesOnly** = 0, **ECpublKeyPartlyCheck** = 1, **ECpublKeyFullCheck** = 2, **PublKeyChecingOffMode**, **EC_PublKeyCheckModeLast** = 0x7FFFFFFF }
- enum **CCEcpkiScaProtection_t** { **SCAP_Inactive**, **SCAP_Active**, **SCAP_OFF_MODE**, **SCAP_LAST** = 0x7FFFFFFF }

2.8.6.4.6 Macro Definition Documentation

#define CC_ECPKI_FIPS_ORDER_LENGTH (256/**CC_BITS_IN_BYTE**)

The order length for FIPS ECC tests.

#define CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS (10 + 3***CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**)

The size of the internal buffer in words.

2.8.6.4.7 Typedef Documentation

typedef struct **CCEcdhFipsKatContext_t CCEcdhFipsKatContext_t**

ECDH KAT data structures for FIPS certification.

typedef struct **CCEcdhTempData_t CCEcdhTempData_t**

The type of the ECDH temporary data.

typedef struct **CCEcdsaFipsKatContext_t CCEcdsaFipsKatContext_t**

ECDSA KAT data structures for FIPS certification. The ECDSA KAT tests are defined for domain 256r1.

typedef uint32_t CCEcdsaSignIntBuff_t[**CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS**]

The internal buffer used in the signing process.

typedef struct **CCEcdsaSignUserContext_t CCEcdsaSignUserContext_t**

This context saves the state of the operation, and must be saved by the user until the end of the API flow.

typedef uint32_t CCEcdsaVerifyIntBuff_t[**CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS**]

The internal buffer used in the verification process.

typedef struct **CCEcdsaVerifyUserContext_t CCEcdsaVerifyUserContext_t**

The context saves the state of the operation, and must be saved by the user until the end of the API flow.

typedef struct **CCEciesTempData_t CCEciesTempData_t**

The temporary data definition of the ECIES.

typedef struct **CCEcpkiBuildTempData_t CCEcpkiBuildTempData_t**

EC build temporary data.

typedef struct **CCEcpkiKgFipsContext_t CCEcpkiKgFipsContext_t**

ECPKI data structures for FIPS certification.

typedef struct **CCEcpkiKgTempData_t CCEcpkiKgTempData_t**

The temporary data type of the ECPKI KG.

typedef struct **CCEcpkiUserPrivKey_t CCEcpkiUserPrivKey_t**

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaSign().

typedef struct **CCEcpkiUserPublKey_t CCEcpkiUserPublKey_t**

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaVerify().

2.8.6.4.8 Enumeration Type Documentation

enum **CCEcpkiDomainID_t**

For more information, see *SECG SEC2 Recommended Elliptic Curve Domain Parameters*, Version 1.0.

Enumerator:

Enum	Description
`CC_ECPKI_DomainID_secp192k1`	EC secp192k1.
`CC_ECPKI_DomainID_secp192r1`	EC secp192r1.
`CC_ECPKI_DomainID_secp224k1`	EC secp224k1.
`CC_ECPKI_DomainID_secp224r1`	EC secp224r1.
`CC_ECPKI_DomainID_secp256k1`	EC secp256k1.
`CC_ECPKI_DomainID_secp256r1`	EC secp256r1.
`CC_ECPKI_DomainID_secp384r1`	EC secp384r1.
`CC_ECPKI_DomainID_secp521r1`	EC secp521r1.
`CC_ECPKI_DomainID_OffMode`	Reserved.

Enum	Description
`CC_ECPKI_DomainIDLast`	Reserved.

enum **CCEcpkiHashOpMode_t**

Defines hash modes according to *IEEE 1363-2000 IEEE Standard for Standard Specifications for Public-Key Cryptography*.

Enumerator:

Enum	Description
`CC_ECPKI_HASH_SHA1_mode`	The message data will be hashed with SHA-1.
`CC_ECPKI_HASH_SHA224_mode`	The message data will be hashed with SHA-224.
`CC_ECPKI_HASH_SHA256_mode`	The message data will be hashed with SHA-256.
`CC_ECPKI_HASH_SHA384_mode`	The message data will be hashed with SHA-384.
`CC_ECPKI_HASH_SHA512_mode`	The message data will be hashed with SHA-512.
`CC_ECPKI_AFTER_HASH_SHA1_mode`	The message data is a digest of SHA-1 and will not be hashed.
`CC_ECPKI_AFTER_HASH_SHA224_mode`	The message data is a digest of SHA-224 and will not be hashed.
`CC_ECPKI_AFTER_HASH_SHA256_mode`	The message data is a digest of SHA-256 and will not be hashed.
`CC_ECPKI_AFTER_HASH_SHA384_mode`	The message data is a digest of SHA-384 and will not be hashed.
`CC_ECPKI_AFTER_HASH_SHA512_mode`	The message data is a digest of SHA-512 and will not be hashed.
`CC_ECPKI_HASH_NumOfModes`	The maximal number of hash modes.
`CC_ECPKI_HASH_OpModeLast`	Reserved.

enum **CCEcpkiPointCompression_t**

EC point-compression identifiers.

Enumerator:

Enum	Description
`CC_EC_PointCompressed`	A compressed point.
`CC_EC_PointUncompressed`	An uncompressed point.
`CC_EC_PointContWrong`	An incorrect point-control value.
`CC_EC_PointHybrid`	A hybrid point.
`CC_EC_PointCompresOffMode`	Reserved.
`CC_ECPKI_PointCompressionLast`	Reserved.

enum **CCEcpkiScaProtection_t**

SW SCA protection type.

Enumerator:

Enum	Description
`SCAP_Inactive`	SCA protection inactive.
`SCAP_Active`	SCA protection active.
`SCAP_OFF_MODE`	Reserved.
`SCAP_LAST`	Reserved.

enum **ECPublKeyCheckMode_t**

EC key checks.

Enumerator:

Enum	Description
`CheckPointersAndSizesOnly`	Check only preliminary input parameters.
`ECpublKeyPartlyCheck`	Check preliminary input parameters and verify that the EC public-key point is on the curve.
`ECpublKeyFullCheck`	Check preliminary input parameters, verify that the EC public-key point is on the curve, and verify that EC_GeneratorOrder*PubKey = 0
`PublKeyChecingOffMode`	Reserved.
`EC_PublKeyCheckModeLast`	Reserved.

2.9 CryptoCell external DMA APIs

Contains all CryptoCell external DMA API definitions.

2.9.1 Modules

- **CryptoCell AES external DMA APIs**

  Contains CryptoCell AES external DMA API definitions.

- **CryptoCell ChaCha external DMA APIs**

  Contains CryptoCell ChaCha external DMA APIs.

- **CryptoCell hash external DMA APIs**

  Contains CryptoCell hash external DMA APIs.

- **Specific errors of the CryptoCell external DMA APIs**

  Contains the CryptoCell external DMA-API error definitions.

2.9.2 CryptoCell AES external DMA APIs

Contains CryptoCell AES external DMA API definitions.

2.9.2.1 Files

- file **mbedtls_aes_ext_dma.h**

  This file contains all the CryptoCell AES external DMA APIs, their enums and definitions.

2.9.2.2 Functions

- int **mbedtls_aes_ext_dma_init** (unsigned int keybits, int encryptDecryptFlag, **CCAesOperationMode_t** operationMode)

  This function initializes the external DMA Control. It configures the AES mode, the direction (encryption or decryption), and the data size.

- int **mbedtls_aes_ext_dma_set_key** (**CCAesOperationMode_t** operationMode, const unsigned char *key, unsigned int keybits)

  This function configures the key.

- int **mbedtls_aes_ext_dma_set_iv** (**CCAesOperationMode_t** operationMode, unsigned char *iv, unsigned int iv_size)

  This function configures the IV.

- int **mbedtls_aes_ext_dma_set_data_size** (uint32_t dataSize, **CCAesOperationMode_t** operationMode)

  This function configures data size which will be written to external DMA interface.

- int **mbedtls_aes_ext_dma_finish** (**CCAesOperationMode_t** operationMode, unsigned char *iv, unsigned int iv_size)

  This function returns the IV after an AES CMAC or a CBCMAC operation.

2.9.2.3 Function Documentation

2.9.2.3.1 int mbedtls_aes_ext_dma_finish (CCAesOperationMode_t operationMode, unsigned char * iv, unsigned int iv_size)

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The AES mode. Supported modes are: ECB, CBC, CTR, CBC_MAC, CMAC or OFB.
out	`iv`	The AES IV buffer.
in	`iv_size`	The size of the IV. Must be 16 bytes.

2.9.2.3.2 int mbedtls_aes_ext_dma_init (unsigned int keybits, int encryptDecryptFlag, CCAesOperationMode_t operationMode)

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`keybits`	AES key size. Valid values are: 128 bits, 192 bits, or 256 bits.
in	`encryptDecryptFlag`	0: Encrypt. 1: Decrypt.
in	`operationMode`	AES mode. Supported modes are: ECB, CBC, CTR, CBC_MAC, CMAC, or OFB.

2.9.2.3.3 int mbedtls_aes_ext_dma_set_data_size (uint32_t dataSize, CCAesOperationMode_t operationMode)

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`dataSize`	Size of input data in bytes.
in	`operationMode`	The AES mode. Supported modes are: ECB, CBC, CTR, CBC_MAC, CMAC or OFB.

2.9.2.3.4 int mbedtls_aes_ext_dma_set_iv (CCAesOperationMode_t operationMode, unsigned char * iv, unsigned int iv_size)

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	AES mode. Supported modes are: ECB, CBC, CTR, CBC_MAC, CMAC or OFB.
in	`iv`	The AES IV buffer.
in	`iv_size`	The size of the IV. Must be 16 bytes.

2.9.2.3.5 int mbedtls_aes_ext_dma_set_key (CCAesOperationMode_t operationMode, const unsigned char * key, unsigned int keybits)

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	AES mode. Supported modes are: ECB, CBC, CTR, CBC_MAC, CMAC or OFB.
in	`key`	The AES key buffer.
in	`keybits`	The size of the AES Key. Valid values are: 128 bits, 192 bits, or 256 bits.

2.9.3 CryptoCell ChaCha external DMA APIs

Contains CryptoCell ChaCha external DMA APIs.

2.9.3.1 Files

- file **mbedtls_chacha_ext_dma.h**

  This file contains all the CryptoCell ChaCha external DMA APIs, their enums and definitions.

2.9.3.2 Functions

- int **mbedtls_ext_dma_chacha_init** (uint8_t *pNonce, mbedtls_chacha_nonce_size_t nonceSizeFlag, uint8_t *pKey, uint32_t keySizebytes, uint32_t initialCounter, mbedtls_chacha_encrypt_mode_t EncryptDecryptFlag, uint32_t dataSize)

  This function initializes the external DMA control.

- int **mbedtls_chacha_ext_dma_finish** (void)

  This function frees used resources.

2.9.3.3 Function Documentation

2.9.3.3.1 int mbedtls_chacha_ext_dma_finish (void)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_ext_dma_error.h** on failure.

2.9.3.3.2 int mbedtls_ext_dma_chacha_init (uint8_t * pNonce, mbedtls_chacha_nonce_size_t nonceSizeFlag, uint8_t * pKey, uint32_t keySizebytes, uint32_t initialCounter, mbedtls_chacha_encrypt_mode_t EncryptDecryptFlag, uint32_t dataSize)

It configures the ChaCha mode, the initial hash value, and other configurations in the ChaCha engine.

Returns:

0 on success.

A non-zero value from **mbedtls_ext_dma_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`pNonce`	The nonce buffer.
in	`nonceSizeFlag`	The nonce size flag.
in	`pKey`	The key buffer.
in	`keySizebytes`	The size of the key buffer. Must be 32 bytes.
in	`initialCounter`	Initial counter value.
in	`EncryptDecryptFlag`	The ChaCha operation: Encrypt or Decrypt.
in	`dataSize`	Input data length in bytes

2.9.4 CryptoCell hash external DMA APIs

Contains CryptoCell hash external DMA APIs.

2.9.4.1 Files

- file **mbedtls_hash_ext_dma.h**

  This file contains all the CryptoCell hash external DMA APIs, their enums and definitions.

2.9.4.2 Functions

- int **mbedtls_hash_ext_dma_init** (**CCHashOperationMode_t** operationMode, uint32_t dataSize)

  This function initializes the External DMA Control.

- int **mbedtls_hash_ext_dma_finish** (**CCHashOperationMode_t** operationMode, uint32_t digestBufferSize, uint32_t *digestBuffer)

  This function returns the digest after the hash operation, and frees used resources.

2.9.4.3 Function Documentation

2.9.4.3.1 int mbedtls_hash_ext_dma_finish (CCHashOperationMode_t operationMode, uint32_t digestBufferSize, uint32_t * digestBuffer)

Returns:

CC_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The hash mode. Supported modes are: SHA1, SHA224 or SHA256.
in	`digestBufferSize`	The size of the hash digest in bytes.
out	`digestBuffer`	The output digest buffer.

2.9.4.3.2 int mbedtls_hash_ext_dma_init (CCHashOperationMode_t operationMode, uint32_t dataSize)

It configures the hash mode, the initial hash value, and other configurations.

Returns:

CC_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The hash mode. Supported modes are: SHA1, SHA224 or SHA256.
in	`dataSize`	Input data size in bytes.

2.9.5 Specific errors of the CryptoCell external DMA APIs

Contains the CryptoCell external DMA-API error definitions.

2.9.5.1 Files

- file **mbedtls_ext_dma_error.h**

  This file contains the error definitions of the CryptoCell external DMA APIs.

2.9.5.2 Macros

- #define **EXT_DMA_AES_ILLEGAL_OPERATION_MODE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x00UL)
- #define **EXT_DMA_AES_INVALID_ENCRYPT_MODE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x01UL)
- #define **EXT_DMA_AES_DECRYPTION_NOT_ALLOWED_ON_THIS_MODE** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x02UL)
- #define **EXT_DMA_AES_ILLEGAL_KEY_SIZE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x03UL)

- #define **EXT_DMA_AES_INVALID_IV_OR_TWEAK_PTR_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x04UL)
- #define **EXT_DMA_HASH_ILLEGAL_OPERATION_MODE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x05UL)
- #define **EXT_DMA_HASH_INVALID_RESULT_BUFFER_POINTER_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x06UL)
- #define **EXT_DMA_HASH_ILLEGAL_PARAMS_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x07UL)
- #define **EXT_DMA_CHACHA_INVALID_NONCE_PTR_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x08UL)
- #define **EXT_DMA_CHACHA_INVALID_ENCRYPT_MODE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0x09UL)
- #define **EXT_DMA_CHACHA_INVALID_KEY_POINTER_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0xAUL)
- #define **EXT_DMA_CHACHA_ILLEGAL_KEY_SIZE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0xBUL)
- #define **EXT_DMA_CHACHA_INVALID_NONCE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0xCUL)
- #define **EXT_DMA_CHACHA_ILLEGAL_INPUT_SIZE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0xDUL)
- #define **EXT_DMA_ILLEGAL_INPUT_SIZE_ERROR** (**CC_EXT_DMA_MODULE_ERROR_BASE** + 0xF0UL)

2.9.5.3 Macro Definition Documentation

2.9.5.3.1 #define EXT_DMA_AES_DECRYPTION_NOT_ALLOWED_ON_THIS_MODE (CC_EXT_DMA_MODULE_ERROR_BASE + 0x02UL)

Illegal decryption mode.

2.9.5.3.2 #define EXT_DMA_AES_ILLEGAL_KEY_SIZE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x03UL)

Illegal key size.

2.9.5.3.3 #define EXT_DMA_AES_ILLEGAL_OPERATION_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x00UL)

Illegal mode.

2.9.5.3.4 #define EXT_DMA_AES_INVALID_ENCRYPT_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x01UL)

Illegal encryption mode.

2.9.5.3.5 #define EXT_DMA_AES_INVALID_IV_OR_TWEAK_PTR_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x04UL)

Illegal IV.

2.9.5.3.6 #define EXT_DMA_CHACHA_ILLEGAL_INPUT_SIZE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xDUL)

Illegal input size.

2.9.5.3.7 #define EXT_DMA_CHACHA_ILLEGAL_KEY_SIZE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xBUL)

Invalid key size.

2.9.5.3.8 #define EXT_DMA_CHACHA_INVALID_ENCRYPT_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x09UL)

Invalid encrypt or decrypt mode.

2.9.5.3.9 #define EXT_DMA_CHACHA_INVALID_KEY_POINTER_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xAUL)

Invalid key pointer.

2.9.5.3.10 #define EXT_DMA_CHACHA_INVALID_NONCE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xCUL)

Invalid nonce size flag.

2.9.5.3.11 #define EXT_DMA_CHACHA_INVALID_NONCE_PTR_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x08UL)

Invalid nonce.

2.9.5.3.12 #define EXT_DMA_HASH_ILLEGAL_OPERATION_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x05UL)

Illegal hash operation mode.

2.9.5.3.13 #define EXT_DMA_HASH_ILLEGAL_PARAMS_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x07UL)

Illegal parameters.

2.9.5.3.14 #define EXT_DMA_HASH_INVALID_RESULT_BUFFER_POINTER_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x06UL)

Illegal result buffer.

2.9.5.3.15 #define EXT_DMA_ILLEGAL_INPUT_SIZE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xF0UL)

Illegal input size.

2.10 CryptoCell hash APIs

Contains all CryptoCell hash APIs and definitions.

2.10.1 Modules

- **CryptoCell-312 hardware limitations for hash**
- **Typical usage of hash in CryptoCell-312**
- **CryptoCell SHA-512 truncated APIs**

  Contains all CryptoCell SHA-512 truncated APIs.
- **CryptoCell hash API definitions**

  Contains CryptoCell hash API definitions.
- **CryptoCell hash API project-specific definitions**

  Contains the project-specific hash API definitions.

2.10.2 Detailed description

The hash or Message Digest (MD) module allows you to calculate hash digests from data, and create signatures based on those hash digests.

HMAC is a wrapping algorithm that uses one of the supported hash algorithms and a key, to generate a unique authentication code over the input data.

All hash algorithms can be accessed via the generic MD layer. For more information, see **mbedtls_md_setup()**.

For more information on supported hash algorithms,

See also:

CryptoCell-312 hardware limitations for hash.

For the implementation of hash and HMAC, see **md.h**.

2.10.3 CryptoCell-312 hardware limitations for hash

The CryptoCell-312 hardware supports accelerated hash operations for the following modes:

- SHA-1
- SHA-224
- SHA-256

  SHA-384 and SHA-512 operations are only supported in a non-accelerated software mode.

To support the accelerated algorithms, the following conditions must be met:

- The input buffer must be DMA-able.
- The input buffer must be physically contingous block in memory.
- Buffer size must be up to 64KB.
- The context must also be DMA-able, as partial and final results are written to the context.

2.10.4 Typical usage of hash in CryptoCell-312

The following is a typical hash Block operation flow directly using the SHA module:

1. **mbedtls_sha1_init()**.
2. **mbedtls_sha1_starts_ret()**.
3. **mbedtls_sha1_update_ret()**.
4. **mbedtls_sha1_finish_ret()**.

The following is a typical HMAC Block operation flow using the MD module:

5. **mbedtls_md_setup()**.
6. **mbedtls_md_hmac_starts()**.
7. **mbedtls_md_hmac_update()**.
8. **mbedtls_md_hmac_finish()**.

2.10.5 CryptoCell SHA-512 truncated APIs

Contains all CryptoCell SHA-512 truncated APIs.

2.10.5.1 Files

- file **mbedtls_cc_sha512_t.h**

  This file contains all of the CryptoCell SHA-512 truncated APIs, their enums and definitions.

2.10.5.2 Functions

- void **mbedtls_sha512_t_init** (**mbedtls_sha512_context** *ctx)

  This function initializes the SHA-512_t context.

- void **mbedtls_sha512_t_free** (**mbedtls_sha512_context** *ctx)

  This function clears the SHA-512_t context.

- void **mbedtls_sha512_t_starts** (**mbedtls_sha512_context** *ctx, int is224)

  This function starts a SHA-512_t checksum calculation.

- void **mbedtls_sha512_t_update** (**mbedtls_sha512_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512_t checksum calculation.

- void **mbedtls_sha512_t_finish** (**mbedtls_sha512_context** *ctx, unsigned char output[32], int is224)

  This function finishes the SHA-512_t operation, and writes the result to the output buffer.

- void **mbedtls_sha512_t** (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-512 checksum of a buffer.

2.10.5.3 Function Documentation

2.10.5.3.1 void mbedtls_sha512_t (const unsigned char * input, size_t ilen, unsigned char output[32], int is224)

The function performs the following operations:

- o Allocates the context.
- o Calculates the checksum.
- o Frees the context.

The SHA-512 result is calculated as output = SHA-512(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The SHA-512/256 or SHA-512/224 checksum result.
`is224`	Determines which function to use: 0: Use SHA-512/256, or 1: Use SHA-512/224.

2.10.5.3.2 void mbedtls_sha512_t_finish (mbedtls_sha512_context * ctx, unsigned char output[32], int is224)

- For SHA512/224, the output buffer will include the 28 leftmost bytes of the SHA-512 digest.
- For SHA512/256, the output buffer will include the 32 leftmost bytes of the SHA-512 digest.

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context.
`output`	The SHA-512/256 or SHA-512/224 checksum result.
`is224`	Determines which function to use: 0: Use SHA-512/256, or 1: Use SHA-512/224.

2.10.5.3.3 void mbedtls_sha512_t_free (mbedtls_sha512_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context to clear.

2.10.5.3.4 void mbedtls_sha512_t_init (mbedtls_sha512_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context to initialize.

2.10.5.3.5 void mbedtls_sha512_t_starts (mbedtls_sha512_context * ctx, int is224)

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context to initialize.
`is224`	Determines which function to use: 0: Use SHA-512/256, or 1: Use SHA-512/224.

2.10.5.3.6 void mbedtls_sha512_t_update (mbedtls_sha512_context * ctx, const unsigned char * input, size_t ilen)

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

2.10.6 CryptoCell hash API definitions

Contains CryptoCell hash API definitions.

2.10.6.1 Files

- file **cc_hash_defs.h**

  This file contains definitions of the CryptoCell hash APIs.

2.10.6.2 Data Structures

- struct **CCHashUserContext_t**

2.10.6.3 Macros

- #define **CC_HASH_RESULT_SIZE_IN_WORDS** 16
- #define **CC_HASH_MD5_DIGEST_SIZE_IN_BYTES** 16
- #define **CC_HASH_MD5_DIGEST_SIZE_IN_WORDS** 4
- #define **CC_HASH_SHA1_DIGEST_SIZE_IN_BYTES** 20
- #define **CC_HASH_SHA1_DIGEST_SIZE_IN_WORDS** 5
- #define **CC_HASH_SHA224_DIGEST_SIZE_IN_WORDS** 7
- #define **CC_HASH_SHA256_DIGEST_SIZE_IN_WORDS** 8
- #define **CC_HASH_SHA384_DIGEST_SIZE_IN_WORDS** 12
- #define **CC_HASH_SHA512_DIGEST_SIZE_IN_WORDS** 16
- #define **CC_HASH_SHA224_DIGEST_SIZE_IN_BYTES** 28
- #define **CC_HASH_SHA256_DIGEST_SIZE_IN_BYTES** 32
- #define **CC_HASH_SHA384_DIGEST_SIZE_IN_BYTES** 48
- #define **CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES** 64
- #define **CC_HASH_BLOCK_SIZE_IN_WORDS** 16
- #define **CC_HASH_BLOCK_SIZE_IN_BYTES** 64
- #define **CC_HASH_SHA512_BLOCK_SIZE_IN_WORDS** 32
- #define **CC_HASH_SHA512_BLOCK_SIZE_IN_BYTES** 128
- #define **CC_HASH_UPDATE_DATA_MAX_SIZE_IN_BYTES** (1 << 29)

2.10.6.4 Typedefs

- typedef uint32_t **CCHashResultBuf_t**[**CC_HASH_RESULT_SIZE_IN_WORDS**]
- typedef struct **CCHashUserContext_t CCHashUserContext_t**

2.10.6.5 Enumerations

- enum **CCHashOperationMode_t** { **CC_HASH_SHA1_mode** = 0, **CC_HASH_SHA224_mode** = 1, **CC_HASH_SHA256_mode** = 2, **CC_HASH_SHA384_mode** = 3, **CC_HASH_SHA512_mode** = 4, **CC_HASH_MD5_mode** = 5, **CC_HASH_NumOfModes**, **CC_HASH_OperationModeLast** = 0x7FFFFFFF }

2.10.6.6 Macro Definition Documentation

2.10.6.6.1 #define CC_HASH_BLOCK_SIZE_IN_BYTES 64

The size of the SHA-1 hash block in bytes.

2.10.6.6.2 #define CC_HASH_BLOCK_SIZE_IN_WORDS 16

The size of the SHA-1 hash block in words.

2.10.6.6.3 #define CC_HASH_MD5_DIGEST_SIZE_IN_BYTES 16

The size of the MD5 digest result in bytes.

2.10.6.6.4 #define CC_HASH_MD5_DIGEST_SIZE_IN_WORDS 4

The size of the MD5 digest result in words.

2.10.6.6.5 #define CC_HASH_RESULT_SIZE_IN_WORDS 16

The size of the hash result in words. The maximal size for SHA-512 is 512 bits.

2.10.6.6.6 #define CC_HASH_SHA1_DIGEST_SIZE_IN_BYTES 20

The size of the SHA-1 digest result in bytes.

2.10.6.6.7 #define CC_HASH_SHA1_DIGEST_SIZE_IN_WORDS 5

The size of the SHA-1 digest result in words.

2.10.6.6.8 #define CC_HASH_SHA224_DIGEST_SIZE_IN_BYTES 28

The size of the SHA-256 digest result in bytes.

2.10.6.6.9 #define CC_HASH_SHA224_DIGEST_SIZE_IN_WORDS 7

The size of the SHA-224 digest result in words.

2.10.6.6.10 #define CC_HASH_SHA256_DIGEST_SIZE_IN_BYTES 32

The size of the SHA-256 digest result in bytes.

2.10.6.6.11 #define CC_HASH_SHA256_DIGEST_SIZE_IN_WORDS 8

The size of the SHA-256 digest result in words.

2.10.6.6.12 #define CC_HASH_SHA384_DIGEST_SIZE_IN_BYTES 48

The size of the SHA-384 digest result in bytes.

2.10.6.6.13 #define CC_HASH_SHA384_DIGEST_SIZE_IN_WORDS 12

The size of the SHA-384 digest result in words.

2.10.6.6.14 #define CC_HASH_SHA512_BLOCK_SIZE_IN_BYTES 128

The size of the SHA-2 hash block in bytes.

2.10.6.6.15 #define CC_HASH_SHA512_BLOCK_SIZE_IN_WORDS 32

The size of the SHA-2 hash block in words.

2.10.6.6.16 #define CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES 64

The size of the SHA-512 digest result in bytes.

2.10.6.6.17 #define CC_HASH_SHA512_DIGEST_SIZE_IN_WORDS 16

The size of the SHA-512 digest result in words.

2.10.6.6.18 #define CC_HASH_UPDATE_DATA_MAX_SIZE_IN_BYTES (1 << 29)

The maximal data size for the update operation.

2.10.6.7 Typedef Documentation

2.10.6.7.1 typedef uint32_t CCHashResultBuf_t[CC_HASH_RESULT_SIZE_IN_WORDS]

The hash result buffer.

2.10.6.7.2 typedef struct CCHashUserContext_t CCHashUserContext_t

The context prototype of the user. The argument type that is passed by the user to the hash APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.10.6.8 Enumeration Type Documentation

2.10.6.8.1 enum CCHashOperationMode_t

The hash operation mode.

Enumerator:

Enum	Description
`CC_HASH_SHA1_mode`	SHA-1.
`CC_HASH_SHA224_mode`	SHA-224.
`CC_HASH_SHA256_mode`	SHA-256.
`CC_HASH_SHA384_mode`	SHA-384.
`CC_HASH_SHA512_mode`	SHA-512.
`CC_HASH_MD5_mode`	MD5.
`CC_HASH_NumOfModes`	The number of hash modes.
`CC_HASH_OperationModeLast`	Reserved.

2.10.7 CryptoCell hash API project-specific definitions

Contains the project-specific hash API definitions.

2.10.7.1 Files

- file **cc_hash_defs_proj.h**

  This file contains the project-specific definitions of hash APIs.

2.10.7.2 Macros

- #define **CC_HASH_USER_CTX_SIZE_IN_WORDS** 60

2.10.7.3 Macro Definition Documentation

2.10.7.3.1 #define CC_HASH_USER_CTX_SIZE_IN_WORDS 60

The size of the context prototype of the user in words. See **CCHashUserContext_t**.

2.11 CryptoCell management APIs

Contains CryptoCell Management APIs.

2.11.1 Modules

- **Specific errors of the CryptoCell Management APIs**

  Contains the CryptoCell management-API error definitions.

2.11.2 Files

- file **mbedtls_cc_mng.h**

  This file contains all CryptoCell Management APIs and definitions.

2.11.3 Data structures

- union **mbedtls_mng_apbc_part**
- union **mbedtls_mng_apbcconfig**

2.11.4 Macros

- #define **CC_MNG_LCS_CM** 0x0
- #define **CC_MNG_LCS_DM** 0x1
- #define **CC_MNG_LCS_SEC_ENABLED** 0x5
- #define **CC_MNG_LCS_RMA** 0x7

2.11.5 typedefs

- typedef union **mbedtls_mng_apbc_part mbedtls_mng_apbc_part**
- typedef union **mbedtls_mng_apbcconfig mbedtls_mng_apbcconfig**

2.11.6 Enumerations

- enum **mbedtls_mng_rmastatus** { **CC_MNG_NON_RMA** = 0, **CC_MNG_PENDING_RMA** = 1, **CC_MNG_ILLEGAL_STATE** = 2, **CC_MNG_RMA** = 3, **CC_MNG_END_OF_RMA_STATUS** = 0x7FFFFFFF }
- enum **mbedtls_mng_keytype** { **CC_MNG_HUK_KEY** = 0, **CC_MNG_RTL_KEY** = 1, **CC_MNG_PROV_KEY** = 2, **CC_MNG_CE_KEY** = 3, **CC_MNG_ICV_PROV_KEY** = 4, **CC_MNG_ICV_CE_KEY** = 5, **CC_MNG_TOTAL_HW_KEYS** = 6, **CC_MNG_END_OF_KEY_TYPE** = 0x7FFFFFFF }
- enum **mbedtls_mng_apbc_parts** { **CC_MNG_APBC_SEC_ID** = 0, **CC_MNG_APBC_PRIV_ID** = 1, **CC_MNG_APBC_INST_ID** = 2, **CC_MNG_APBC_TOTAL_ID** = 3, **CC_MNG_APBC_END_OF_ID** = 0x7FFFFFFF }
- enum **mbedtls_mng_apbc_parts_config** { **CC_MNG_APBC_NO_CHANGE** = 0, **CC_MNG_APBC_ALLOW_0_ALLOWLOCK_0** = 1, **CC_MNG_APBC_ALLOW_0_ALLOWLOCK_1** = 2, **CC_MNG_APBC_ALLOW_1_ALLOWLOCK_0** = 3, **CC_MNG_APBC_ALLOW_1_ALLOWLOCK_1** = 4, **CC_MNG_APBC_TOTAL_PARTS_CONFIG** = 5, **CC_MNG_APBC_END_OF_PARTS_CONFIG** = 0x7FFFFFFF }

2.11.7 Functions

- int **mbedtls_mng_pending_rma_status_get** (uint32_t *rmaStatus)

  This function reads the OTP word of the OEM flags, and returns the OEM RMA flag status: TRUE or FALSE.
- int **mbedtls_mng_hw_version_get** (uint32_t *partNumber, uint32_t *revision)

  This function verifies and returns the CryptoCell HW version.
- int **mbedtls_mng_cc_sec_mode_set** (CCBool_t isSecAccessMode, CCBool_t isSecModeLock)

  This function sets CryptoCell to Secured mode.
- int **mbedtls_mng_cc_priv_mode_set** (CCBool_t isPrivAccessMode, CCBool_t isPrivModeLock)

  This function sets CryptoCell to Privileged mode.
- int **mbedtls_mng_debug_key_set** (**mbedtls_mng_keytype** keyType, uint32_t *pHwKey, size_t keySize)

  This function sets the shadow register of one of the HW Keys when the device is in CM LCS or DM LCS.
- int **mbedtls_mng_gen_config_get** (uint32_t *pOtpWord)

  This function retrieves the general configuration from the OTP. See Arm CryptoCell-312 Software Integrators Manual.

- int **mbedtls_mng_oem_key_lock** (CCBool_t kcpLock, CCBool_t kceLock)

  This function locks the usage of either Kcp, Kce, or both during runtime, in either Secure LCS or RMA LCS.
- int **mbedtls_mng_apbc_config_set** (**mbedtls_mng_apbc_parts_config** securePartCfg, **mbedtls_mng_apbc_parts_config** privPartCfg, **mbedtls_mng_apbc_parts_config** instPartCfg)

  This function sets CryptoCell APB-C into one of the following modes: Secured access mode, Privileged access mode, or Instruction access mode.
- int **mbedtls_mng_apbc_access** (CCBool_t isApbcAccessUsed)

  This function requests usage of, or releases, the APB-C.
- int **mbedtls_mng_suspend** (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-down CryptoCell.
- int **mbedtls_mng_resume** (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-up CryptoCell.

2.11.8 Macro definition documentation

2.11.8.1 #define CC_MNG_LCS_CM 0x0

Chip manufacturer (CM LCS).

2.11.8.2 #define CC_MNG_LCS_DM 0x1

Device manufacturer (DM LCS).

2.11.8.3 #define CC_MNG_LCS_RMA 0x7

RMA (RMA LCS).

2.11.8.4 #define CC_MNG_LCS_SEC_ENABLED 0x5

Security enabled (Secure LCS).

2.11.9 typedef documentation

2.11.9.1 typedef union mbedtls_mng_apbc_part mbedtls_mng_apbc_part

A uint8_t representation for the APB-C parts in the AO_APB_FILTERING register.

2.11.9.2 typedef union mbedtls_mng_apbcconfig mbedtls_mng_apbcconfig

Input to the **mbedtls_mng_apbc_config_set()** function.

2.11.10 Enumeration type documentation

2.11.10.1 enum mbedtls_mng_apbc_parts

APB-C only part IDs.

Enumerator:

Enum	Description
`CC_MNG_APBC_SEC_ID`	Secure accesses.
`CC_MNG_APBC_PRIV_ID`	Privileged accesses.
`CC_MNG_APBC_INST_ID`	Instruction accesses.
`CC_MNG_APBC_TOTAL_ID`	Total part IDs.
`CC_MNG_APBC_END_OF_ID`	Reserved.

2.11.10.2 enum mbedtls_mng_apbc_parts_config

APB-C part configuration.

Enumerator:

Enum	Description
`CC_MNG_APBC_NO_CHANGE`	Use APB-C as an input when there is no need to change bits. Modify bit = 0.
`CC_MNG_APBC_ALLOW_0_ALLOWLOCK_0`	Use APB-C as an input when you need to set the 'Allow' bit to '0' and leave this part unlocked. Modify bit = 1, Allow bit = 0, Allow Lock bit = 0.
`CC_MNG_APBC_ALLOW_0_ALLOWLOCK_1`	Use APB-C as an input when you need to set the 'Allow' bit to '0' and lock this part. Modify bit = 1, Allow bit = 0, Allow Lock bit = 1.
`CC_MNG_APBC_ALLOW_1_ALLOWLOCK_0`	Use APB-C as an input when you need to set the 'Allow' bit to '1' and leave this part unlocked. Modify bit = 1, Allow bit = 1, Allow Lock bit = 0.
`CC_MNG_APBC_ALLOW_1_ALLOWLOCK_1`	Use APB-C as an input when you need to set the 'Allow' bit to '1' and lock this part. Modify bit = 1, Allow bit = 1, Allow Lock bit = 1.
`CC_MNG_APBC_TOTAL_PARTS_CONFIG`	Total parts.
`CC_MNG_APBC_END_OF_PARTS_CONFIG`	Reserved.

2.11.10.3 enum mbedtls_mng_keytype

AES HW key types.

Enumerator:

Enum	Description
`CC_MNG_HUK_KEY`	Device root key (HUK).
`CC_MNG_RTL_KEY`	Platform key (Krtl).
`CC_MNG_PROV_KEY`	ICV provisioning key (Kcp).
`CC_MNG_CE_KEY`	OEM code-encryption key (Kce).
`CC_MNG_ICV_PROV_KEY`	OEM provisioning key (Kpicv).
`CC_MNG_ICV_CE_KEY`	ICV code-encryption key (Kceicv).
`CC_MNG_TOTAL_HW_KEYS`	Total number of HW Keys.
`CC_MNG_END_OF_KEY_TYPE`	Reserved.

2.11.10.4 enum mbedtls_mng_rmastatus

RMA statuses.

Enumerator:

Enum	Description
`CC_MNG_NON_RMA`	Non-RMA: bit [30] = 0, bit [31] = 0.
`CC_MNG_PENDING_RMA`	Pending RMA: bit [30] = 1, bit [31] = 0.
`CC_MNG_ILLEGAL_STATE`	Illegal state: bit [30] = 0, bit [31] = 1.
`CC_MNG_RMA`	RMA: bit [30] = 1, bit [31] = 1.
`CC_MNG_END_OF_RMA_STATUS`	Reserved.

2.11.11 Function documentation

2.11.11.1 int mbedtls_mng_apbc_access (CCBool_t isApbcAccessUsed)

This function must be called before and after each use of APB-C.

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`isApbcAccessUsed`	TRUE: Request usage of APB-C. FALSE: Free APB-C.

2.11.11.2 int mbedtls_mng_apbc_config_set (mbedtls_mng_apbc_parts_config securePartCfg, mbedtls_mng_apbc_parts_config privPartCfg, mbedtls_mng_apbc_parts_config instPartCfg)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

Parameter	Description
`securePartCfg`	Secured access mode.
`privPartCfg`	Privileged access mode.
`instPartCfg`	Instruction access mode.

2.11.11.3 int mbedtls_mng_cc_priv_mode_set (CCBool_t isPrivAccessMode, CCBool_t isPrivModeLock)

Setting CryptoCell to Privileged mode can only be done while CryptoCell is idle.

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`isPrivAccessMode`	True: Set CryptoCell to privileged mode. False: Set CryptoCell to unprivileged mode.
in	`isPrivModeLock`	True: Lock CryptoCell to current mode. False: Do not lock CryptoCell to current mode. Allows calling this function again.

2.11.11.4 int mbedtls_mng_cc_sec_mode_set (CCBool_t isSecAccessMode, CCBool_t isSecModeLock)

Setting CryptoCell to Secured mode can only be done while CryptoCell is idle.

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`isSecAccessMode`	True: Set CryptoCell to Secured mode. False: Set CryptoCell to non-Secured mode.
in	`isSecModeLock`	True: Lock CryptoCell to current mode. False: Do not lock CryptoCell to current mode. Allows calling this function again.

2.11.11.5 int mbedtls_mng_debug_key_set (mbedtls_mng_keytype keyType, uint32_t * pHwKey, size_t keySize)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`keyType`	The type of the HW key. One of the following values: HUK, Kcp, Kce, Kpicv, or Kceicv.
in	`pHwKey`	A pointer to the buffer holding the HW key.
in	`keySize`	The size of the HW key in bytes.

2.11.11.6 int mbedtls_mng_gen_config_get (uint32_t * pOtpWord)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
out	`pOtpWord`	The OTP configuration word.

2.11.11.7 int mbedtls_mng_hw_version_get (uint32_t * partNumber, uint32_t * revision)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
out	`partNumber`	The part number.
out	`revision`	The HW version.

2.11.11.8 int mbedtls_mng_oem_key_lock (CCBool_t kcpLock, CCBool_t kceLock)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`kcpLock`	The flag for locking Kcp usage.
in	`kceLock`	The flag for locking Kce usage.

2.11.11.9 int mbedtls_mng_pending_rma_status_get (uint32_t * rmaStatus)

The function returns the value only in DM LCS or Secure LCS. It validates the device RoT configuration, and returns the value only if both HBK0 and HBK1 are supported. Otherwise, it returns FALSE regardless of the OTP status.

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
out	`rmaStatus`	The RMA status.

2.11.11.10 int mbedtls_mng_resume (uint8_t * pBackupBuffer, size_t backupSize)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`pBackupBuffer`	A pointer to a buffer that can be used for backup.
in	`backupSize`	The size of the backup buffer. Must be at least CC_MNG_MIN_BACKUP_SIZE_IN_BYTES.

2.11.11.11 int mbedtls_mng_suspend (uint8_t * pBackupBuffer, size_t backupSize)

Returns:

CC_OK on success.

A non-zero value from **mbedtls_cc_mng_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`pBackupBuffer`	A pointer to a buffer that can be used for backup.
in	`backupSize`	The size of the backup buffer. Must be at least CC_MNG_MIN_BACKUP_SIZE_IN_BYTES.

2.11.12 Specific errors of the CryptoCell Management APIs

Contains the CryptoCell management-API error definitions.

2.11.12.1 Files

- file **mbedtls_cc_mng_error.h**

  This file contains the error definitions of the CryptoCell management APIs.

2.11.12.2 Macros

- #define **CC_MNG_ILLEGAL_INPUT_PARAM_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x00UL)
- #define **CC_MNG_ILLEGAL_OPERATION_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x01UL)
- #define **CC_MNG_ILLEGAL_PIDR_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x02UL)
- #define **CC_MNG_ILLEGAL_CIDR_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x03UL)
- #define **CC_MNG_APB_SECURE_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x04UL)
- #define **CC_MNG_APB_PRIVILEGE_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x05UL)
- #define **CC_MNG_APBC_SECURE_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x06UL)
- #define **CC_MNG_APBC_PRIVILEGE_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x07UL)
- #define **CC_MNG_APBC_INSTRUCTION_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x08UL)
- #define **CC_MNG_INVALID_KEY_TYPE_ERROR** (**CC_MNG_MODULE_ERROR_BASE** + 0x09UL)
- #define **CC_MNG_ILLEGAL_HUK_SIZE_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0AUL)
- #define **CC_MNG_ILLEGAL_HW_KEY_SIZE_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0BUL)
- #define **CC_MNG_HW_KEY_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0CUL)
- #define **CC_MNG_KCP_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0DUL)
- #define **CC_MNG_KCE_IS_LOCKED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0EUL)

- #define **CC_MNG_RMA_ILLEGAL_STATE_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x0FUL)
- #define **CC_MNG_AO_APB_WRITE_FAILED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x10UL)
- #define **CC_MNG_APBC_ACCESS_FAILED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x11UL)
- #define **CC_MNG_APBC_ACCESS_ALREADY_OFF_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x12UL)
- #define **CC_MNG_APBC_ACCESS_IS_ON_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x13UL)
- #define **CC_MNG_PM_SUSPEND_RESUME_FAILED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x14UL)
- #define **CC_MNG_ILLEGAL_SW_VERSION_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x15UL)
- #define **CC_MNG_HASH_NOT_PROGRAMMED_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x16UL)
- #define **CC_MNG_HBK_ZERO_COUNT_ERR** (**CC_MNG_MODULE_ERROR_BASE** + 0x17UL)

2.11.12.3 Macro Definition Documentation

2.11.12.3.1 #define CC_MNG_AO_APB_WRITE_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x10UL)

Error returned from AO_APB_FILTERING write operation.

2.11.12.3.2 #define CC_MNG_APB_PRIVILEGE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x05UL)

APB Privilege is locked.

2.11.12.3.3 #define CC_MNG_APB_SECURE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x04UL)

APB Secure is locked.

2.11.12.3.4 #define CC_MNG_APBC_ACCESS_ALREADY_OFF_ERR (CC_MNG_MODULE_ERROR_BASE + 0x12UL)

APBC already-off failure.

2.11.12.3.5 #define CC_MNG_APBC_ACCESS_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x11UL)

APBC access failure.

2.11.12.3.6 #define CC_MNG_APBC_ACCESS_IS_ON_ERR (CC_MNG_MODULE_ERROR_BASE + 0x13UL)

APBC access is on failure.

2.11.12.3.7 #define CC_MNG_APBC_INSTRUCTION_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x08UL)

APBC Instruction is locked.

2.11.12.3.8 #define CC_MNG_APBC_PRIVILEGE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x07UL)

APBC Privilege is locked.

2.11.12.3.9 #define CC_MNG_APBC_SECURE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x06UL)

APBC Secure is locked.

2.11.12.3.10 #define CC_MNG_HASH_NOT_PROGRAMMED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x16UL)

Hash Public Key NA.

2.11.12.3.11 #define CC_MNG_HBK_ZERO_COUNT_ERR (CC_MNG_MODULE_ERROR_BASE + 0x17UL)

Illegal hash boot key zero count in the OTP error.

2.11.12.3.12 #define CC_MNG_HW_KEY_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0CUL)

HW key is locked.

2.11.12.3.13 #define CC_MNG_ILLEGAL_CIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x03UL)

Illegal Component ID.

2.11.12.3.14 #define CC_MNG_ILLEGAL_HUK_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0AUL)

Illegal size of HUK.

2.11.12.3.15 #define CC_MNG_ILLEGAL_HW_KEY_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0BUL)

Illegal size for any HW key other than HUK.

2.11.12.3.16 #define CC_MNG_ILLEGAL_INPUT_PARAM_ERR (CC_MNG_MODULE_ERROR_BASE + 0x00UL)

Illegal input parameter.

2.11.12.3.17 #define CC_MNG_ILLEGAL_OPERATION_ERR (CC_MNG_MODULE_ERROR_BASE + 0x01UL)

Illegal operation.

2.11.12.3.18 #define CC_MNG_ILLEGAL_PIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x02UL)

Illegal Peripheral ID.

2.11.12.3.19 #define CC_MNG_ILLEGAL_SW_VERSION_ERR (CC_MNG_MODULE_ERROR_BASE + 0x15UL)

SW version failure.

2.11.12.3.20 #define CC_MNG_INVALID_KEY_TYPE_ERROR (CC_MNG_MODULE_ERROR_BASE + 0x09UL)

Invalid Key type.

2.11.12.3.21 #define CC_MNG_KCE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0EUL)

Kce is locked.

2.11.12.3.22 #define CC_MNG_KCP_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0DUL)

Kcp is locked.

2.11.12.3.23 #define CC_MNG_PM_SUSPEND_RESUME_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x14UL)

PM SUSPEND/RESUME failure.

2.11.12.3.24 #define CC_MNG_RMA_ILLEGAL_STATE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0FUL)

RMA Illegal state.

2.12 CryptoCell PAL APIs

Groups all PAL APIs and definitions.

2.12.1 Modules

- **CryptoCell PAL abort operations**

  Contains CryptoCell PAL abort operations.
- **CryptoCell PAL APB-C APIs**

  Contains PAL APB-C APIs.
- **CryptoCell PAL definitions for Boot Services**

  Contains CryptoCell PAL Secure Boot definitions.
- **CryptoCell PAL entry or exit point APIs**

  Contains PAL initialization and termination APIs.
- **CryptoCell PAL logging APIs and definitions**

  Contains CryptoCell PAL layer log definitions.
- **CryptoCell PAL memory operations**

  Contains memory-operation functions.
- **CryptoCell PAL mutex APIs**

  Contains resource management functions.
- **CryptoCell PAL platform-dependent definitions and types**

  Contains CryptoCell PAL platform-dependent definitions and types.
- **CryptoCell PAL platform-dependent compiler-related definitions**

  Contains CryptoCell PAL platform-dependent compiler-related definitions.
- **CryptoCell PAL power-management APIs**

Contains PAL power-management APIs.

- **CryptoCell PAL TRNG APIs**

  Contains APIs for retrieving TRNG user parameters.

- **Specific errors of the CryptoCell PAL APIs**

  Contains platform-dependent PAL-API error definitions.

2.12.2 CryptoCell PAL abort operations

Contains CryptoCell PAL abort operations.

2.12.2.1 Files

- file **cc_pal_abort.h**

  This file includes all PAL abort APIs.

2.12.2.2 Functions

- void **CC_PalAbort** (const char *exp)

  This function performs the "Abort" operation.

2.12.2.3 Function Documentation

2.12.2.3.1 void CC_PalAbort (const char * exp)

Must be implemented according to platform and OS.

Parameters:

I/O	Parameter	Description
in	`exp`	An optional parameter for a string of chars to indicate the abort operation.

2.12.3 CryptoCell PAL APB-C APIs

Contains PAL APB-C APIs.

2.12.3.1 Files

- file **cc_pal_apbc.h**

  This file contains the definitions and APIs for APB-C implementation.

2.12.3.2 Functions

- void **CC_PalApbcCntrInit** (void)

This function initiates an atomic counter.

- int32_t **CC_PalApbcCntrValue** (void)

  This function returns the number of APB-C access operations.

- CCError_t **CC_PalApbcModeSelect** (**CCBool** isApbcInc)

  This function updates the atomic counter on each call to APB-C access.

2.12.3.3 Function Documentation

2.12.3.3.1 void CC_PalApbcCntrInit (void)

Returns:

Void.

2.12.3.3.2 int32_t CC_PalApbcCntrValue (void)

Returns:

The value of the atomic counter.

2.12.3.3.3 CCError_t CC_PalApbcModeSelect (CCBool isApbcInc)

On each call to APB-C access, the counter is increased. At the end of each operation, the counter is decreased.

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`isApbcInc`	Determines the APB-C mode: TRUE (APB-C start access). FALSE (APB-C finish access).

2.12.4 CryptoCell PAL definitions for Boot Services

Contains CryptoCell PAL Secure Boot definitions.

2.12.4.1 Files

- file **cc_pal_sb_plat.h**

  This file contains platform-dependent definitions used in the Boot Services code.

2.12.4.2 Typedefs

- typedef uint32_t **CCDmaAddr_t**
- typedef uint32_t **CCAddr_t**

2.12.4.3 Typedef Documentation

2.12.4.3.1 typedef uint32_t CCAddr_t

CryptocCell address types: 32 bits or 64 bits, according to platform.

2.12.4.3.2 typedef uint32_t CCDmaAddr_t

DMA address types: 32 bits or 64 bits, according to platform.

2.12.5 CryptoCell PAL entry or exit point APIs

Contains PAL initialization and termination APIs.

2.12.5.1 Files

- file **cc_pal_init.h**

  This file contains the PAL layer entry point.

2.12.5.2 Functions

- int **CC_PalInit** (void)

  This function performs all initializations that may be required by your PAL implementation, specifically by the DMA-able buffer scheme.

- void **CC_PalTerminate** (void)

  This function terminates the PAL implementation and frees the resources that were allocated by **CC_PalInit**.

2.12.5.3 Function Documentation

2.12.5.3.1 int CC_PalInit (void)

It is called by **CC_LibInit**.

The existing implementation allocates a contiguous memory pool that is later used by the CryptoCell implementation. If no initializations are needed in your environment, the function can be minimized to return OK.

Returns:

A non-zero value on failure.

2.12.5.3.2 void CC_PalTerminate (void)

Returns:

Void.

2.12.6 CryptoCell PAL logging APIs and definitions

Contains CryptoCell PAL layer log definitions.

2.12.6.1 Files

- file **cc_pal_log.h**

  This file contains the PAL layer log definitions.

2.12.6.2 Macros

- #define **CC_PAL_LOG_LEVEL_NULL** (-1)
- #define **CC_PAL_LOG_LEVEL_ERR** 0
- #define **CC_PAL_LOG_LEVEL_WARN** 1
- #define **CC_PAL_LOG_LEVEL_INFO** 2
- #define **CC_PAL_LOG_LEVEL_DEBUG** 3
- #define **CC_PAL_LOG_LEVEL_TRACE** 4
- #define **CC_PAL_LOG_LEVEL_DATA** 5
- #define **CC_PAL_LOG_CUR_COMPONENT** 0xFFFFFFFF
- #define **CC_PAL_LOG_CUR_COMPONENT_NAME** "CC"
- #define **CC_PAL_MAX_LOG_LEVEL CC_PAL_LOG_LEVEL_NULL**
- #define **__CC_PAL_LOG_LEVEL_EVAL**(level) level
- #define **_CC_PAL_MAX_LOG_LEVEL** **__CC_PAL_LOG_LEVEL_EVAL**(**CC_PAL_MAX_LOG_LEVEL**)
- #define **_CC_PAL_LOG**(level, format, ...)
- #define **CC_PAL_LOG_ERR**(...) do {} while (0)
- #define **CC_PAL_LOG_WARN**(...) do {} while (0)
- #define **CC_PAL_LOG_INFO**(...) do {} while (0)
- #define **CC_PAL_LOG_DEBUG**(...) do {} while (0)
- #define **CC_PAL_LOG_DUMP_BUF**(msg, buf, size) do {} while (0)
- #define **CC_PAL_LOG_TRACE**(...) do {} while (0)
- #define **CC_PAL_LOG_DATA**(...) do {} while (0)

2.12.6.3 Macro Definition Documentation

2.12.6.3.1 #define __CC_PAL_LOG_LEVEL_EVAL(level) level

Evaluate CC_PAL_MAX_LOG_LEVEL in case provided by caller.

2.12.6.3.2 #define _CC_PAL_LOG(level, format, ...)

```
if (CC_PAL_logMask & CC_PAL_LOG_CUR_COMPONENT) \
        CC_PalLog(CC_PAL_LOG_LEVEL_ ## level, "%s:%s: " format,
CC_PAL_LOG_CUR_COMPONENT_NAME, __func__, ##__VA_ARGS__)
```

Filter logging based on logMask , and dispatch to platform-specific logging mechanism.

2.12.6.3.3 #define _CC_PAL_MAX_LOG_LEVEL __CC_PAL_LOG_LEVEL_EVAL(CC_PAL_MAX_LOG_LEVEL)

The maximal log-level definition.

2.12.6.3.4 #define CC_PAL_LOG_CUR_COMPONENT 0xFFFFFFFF

Default log debugged component.

2.12.6.3.5 #define CC_PAL_LOG_CUR_COMPONENT_NAME "CC"

Default log debugged component.

2.12.6.3.6 #define CC_PAL_LOG_DATA(...) do {} while (0)

Log debug data.

2.12.6.3.7 #define CC_PAL_LOG_DEBUG(...) do {} while (0)

Log debug messages.

2.12.6.3.8 #define CC_PAL_LOG_DUMP_BUF(msg, buf, size) do {} while (0)

Log debug buffer.

2.12.6.3.9 #define CC_PAL_LOG_ERR(...) do {} while (0)

Log messages according to log level.

2.12.6.3.10 #define CC_PAL_LOG_INFO(...) do {} while (0)

Log messages according to log level.

2.12.6.3.11 #define CC_PAL_LOG_LEVEL_DATA 5

PAL log level - data.

2.12.6.3.12 #define CC_PAL_LOG_LEVEL_DEBUG 3

PAL log level - debug.

2.12.6.3.13 #define CC_PAL_LOG_LEVEL_ERR 0

PAL log level - error.

2.12.6.3.14 #define CC_PAL_LOG_LEVEL_INFO 2

PAL log level - info.

2.12.6.3.15 #define CC_PAL_LOG_LEVEL_NULL (-1)

PAL log level - disabled.

2.12.6.3.16 #define CC_PAL_LOG_LEVEL_TRACE 4

PAL log level - trace.

2.12.6.3.17 #define CC_PAL_LOG_LEVEL_WARN 1

PAL log level - warning.

2.12.6.3.18 #define CC_PAL_LOG_TRACE(...) do {} while (0)

Log debug trace.

2.12.6.3.19 #define CC_PAL_LOG_WARN(...) do {} while (0)

Log messages according to log level.

2.12.6.3.20 #define CC_PAL_MAX_LOG_LEVEL CC_PAL_LOG_LEVEL_NULL

Default debug log level, when debug is set to off.

2.12.7 CryptoCell PAL memory operations

Contains memory-operation functions.

2.12.7.1 Modules

- **CryptoCell PAL memory Barrier APIs**

  Contains memory-barrier implementation definitions and APIs.

- **CryptoCell PAL memory mapping APIs**

  Contains memory mapping functions.

2.12.7.2 Files

- file **cc_pal_mem.h**

  This file contains functions for memory operations.

2.12.7.3 Macros

- #define **CC_PalMemCmp**(aTarget, aSource, aSize) CC_PalMemCmpPlat(aTarget, aSource, aSize)

  This function compares between two given buffers, according to the given size.

- #define **CC_PalMemCopy**(aDestination, aSource, aSize) CC_PalMemCopyPlat(aDestination, aSource, aSize)

  This function copies aSize bytes from the source buffer to the destination buffer.

- #define **CC_PalMemMove**(aDestination, aSource, aSize) CC_PalMemMovePlat(aDestination, aSource, aSize)

  This function moves aSize bytes from the source buffer to the destination buffer.

- #define **CC_PalMemSet**(aTarget, aChar, aSize) CC_PalMemSetPlat(aTarget, aChar, aSize)

  This function sets aSize bytes of aChar in the given buffer.

- #define **CC_PalMemSetZero**(aTarget, aSize) CC_PalMemSetZeroPlat(aTarget, aSize)

  This function sets aSize bytes in the given buffer to zeroes.

- #define **CC_PalMemMalloc**(aSize) CC_PalMemMallocPlat(aSize)

  This function allocates a memory buffer according to aSize.

- #define **CC_PalMemRealloc**(aBuffer, aNewSize) CC_PalMemReallocPlat(aBuffer, aNewSize)

  This function reallocates a memory buffer according to aNewSize. The content of the old buffer is moved to the new location.

- #define **CC_PalMemFree**(aBuffer) CC_PalMemFreePlat(aBuffer)

  This function frees a previously-allocated buffer.

2.12.7.4 Macro Definition Documentation

2.12.7.4.1 #define CC_PalMemCmp(aTarget, aSource, aSize) CC_PalMemCmpPlat(aTarget, aSource, aSize)

Returns:

The return values are according to operating-system return values.

2.12.7.4.2 #define CC_PalMemCopy(aDestination, aSource, aSize) CC_PalMemCopyPlat(aDestination, aSource, aSize)

Returns:

void.

2.12.7.4.3 #define CC_PalMemFree(aBuffer) CC_PalMemFreePlat(aBuffer)

Returns:

void.

2.12.7.4.4 #define CC_PalMemMalloc(aSize) CC_PalMemMallocPlat(aSize)

Returns:

A pointer to the allocated buffer on success.

NULL on failure.

2.12.7.4.5 #define CC_PalMemMove(aDestination, aSource, aSize) CC_PalMemMovePlat(aDestination, aSource, aSize)

This function supports overlapped buffers.

Returns:

void.

2.12.7.4.6 #define CC_PalMemRealloc(aBuffer, aNewSize) CC_PalMemReallocPlat(aBuffer, aNewSize)

Returns:

A pointer to the newly-allocated buffer on success.

NULL on failure.

2.12.7.4.7 #define CC_PalMemSet(aTarget, aChar, aSize) CC_PalMemSetPlat(aTarget, aChar, aSize)

Returns:

void.

2.12.7.4.8 #define CC_PalMemSetZero(aTarget, aSize) CC_PalMemSetZeroPlat(aTarget, aSize)

Returns:

void.

2.12.8 CryptoCell PAL memory Barrier APIs

Contains memory-barrier implementation definitions and APIs.

2.12.8.1 Files

- file **cc_pal_barrier.h**

  This file contains the definitions and APIs for memory-barrier implementation.

2.12.8.2 Functions

- void **CC_PalWmb** (void)
- void **CC_PalRmb** (void)

2.12.8.3 Function Documentation

2.12.8.3.1 void CC_PalRmb (void)

This macro puts the memory barrier before the read operation.

Returns:

None

2.12.8.3.2 void CC_PalWmb (void)

This macro puts the memory barrier after the write operation.

Returns:

None

2.12.9 CryptoCell PAL memory mapping APIs

Contains memory mapping functions.

2.12.9.1 Files

- file **cc_pal_memmap.h**

  This file contains functions for memory mapping.

2.12.9.2 Functions

- uint32_t **CC_PalMemMap** (**CCDmaAddr_t** physicalAddress, uint32_t mapSize, uint32_t **ppVirtBuffAddr)

  This function returns the base virtual address that maps the base physical address.

- uint32_t **CC_PalMemUnMap** (uint32_t *pVirtBuffAddr, uint32_t mapSize)

  This function unmaps a specified address range that was previously mapped by **CC_PalMemMap**.

2.12.9.3 Function Documentation

2.12.9.3.1 uint32_t CC_PalMemMap (CCDmaAddr_t physicalAddress, uint32_t mapSize, uint32_t ** ppVirtBuffAddr)

Returns:

0 on success.

A non-zero value in case of failure.

Parameters:

I/O	Parameter	Description
in	`physicalAddress`	The starting physical address of the I/O range to be mapped.
in	`mapSize`	The number of bytes that were mapped.
out	`ppVirtBuffAddr`	A pointer to the base virtual address to which the physical pages were mapped.

2.12.9.3.2 uint32_t CC_PalMemUnMap (uint32_t * pVirtBuffAddr, uint32_t mapSize)

Returns:

0 on success.

A non-zero value in case of failure.

Parameters:

I/O	Parameter	Description
in	`pVirtBuffAddr`	A pointer to the base virtual address to which the physical pages were mapped.
in	`mapSize`	The number of bytes that were mapped.

2.12.10 CryptoCell PAL mutex APIs

Contains resource management functions.

2.12.10.1 Files

- file **cc_pal_mutex.h**

  This file contains functions for resource management (mutex operations).

2.12.10.2 Functions

- CCError_t **CC_PalMutexCreate** (CC_PalMutex *pMutexId)

  This function creates a mutex.

- CCError_t **CC_PalMutexDestroy** (CC_PalMutex *pMutexId)

  This function destroys a mutex.

- CCError_t **CC_PalMutexLock** (CC_PalMutex *pMutexId, uint32_t aTimeOut)

  This function waits for a mutex with aTimeOut.

- CCError_t **CC_PalMutexUnlock** (CC_PalMutex *pMutexId)

  This function releases the mutex.

2.12.10.3 Function Documentation

2.12.10.3.1 CCError_t CC_PalMutexCreate (CC_PalMutex * pMutexId)

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
out	`pMutexId`	A pointer to the handle of the created mutex.

2.12.10.3.2 CCError_t CC_PalMutexDestroy (CC_PalMutex * pMutexId)

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to handle of the mutex to destroy.

2.12.10.3.3 CCError_t CC_PalMutexLock (CC_PalMutex * pMutexId, uint32_t aTimeOut)

aTimeOut is specified in milliseconds. A value of aTimeOut=CC_INFINITE means that the function will not return.

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to handle of the mutex.
in	`aTimeOut`	The timeout in mSec, or CC_INFINITE.

2.12.10.3.4 CCError_t CC_PalMutexUnlock (CC_PalMutex * pMutexId)

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to the handle of the mutex.

2.12.11 CryptoCell PAL platform-dependent definitions and types

Contains CryptoCell PAL platform-dependent definitions and types.

2.12.11.1 Files

- file **cc_pal_types.h**

  This file contains definitions and types of CryptoCell PAL platform-dependent APIs.

2.12.11.2 Macros

- #define **CC_SUCCESS** 0UL
- #define **CC_FAIL** 1UL
- #define **CC_OK** 0
- #define **CC_UNUSED_PARAM**(prm) ((void)prm)
- #define **CC_MAX_UINT32_VAL** (0xFFFFFFFF)
- #define **CC_MIN**(a, b) min(a , b)
- #define **CC_MAX**(a, b) max(a , b)
- #define **CALC_FULL_BYTES**(numBits) ((numBits)/**CC_BITS_IN_BYTE** + (((numBits) & (**CC_BITS_IN_BYTE**-1)) > 0))
- #define **CALC_FULL_32BIT_WORDS**(numBits) ((numBits)/**CC_BITS_IN_32BIT_WORD** + (((numBits) & (**CC_BITS_IN_32BIT_WORD**-1)) > 0))
- #define **CALC_32BIT_WORDS_FROM_BYTES**(sizebytes) ((sizebytes)/**CC_32BIT_WORD_SIZE** + (((sizebytes) & (**CC_32BIT_WORD_SIZE**-1)) > 0))
- #define **CALC_32BIT_WORDS_FROM_64BIT_DWORD**(sizeWords) (sizeWords ***CC_32BIT_WORD_IN_64BIT_DWORD**)
- #define **ROUNDUP_BITS_TO_32BIT_WORD**(numBits) (**CALC_FULL_32BIT_WORDS**(numBits) ***CC_BITS_IN_32BIT_WORD**)
- #define **ROUNDUP_BITS_TO_BYTES**(numBits) (**CALC_FULL_BYTES**(numBits) ***CC_BITS_IN_BYTE**)

- #define **ROUNDUP_BYTES_TO_32BIT_WORD**(sizebytes) (**CALC_32BIT_WORDS_FROM_BYTES**(sizebytes) ***CC_32BIT_WORD_SIZE**)
- #define **CC_1K_SIZE_IN_BYTES** 1024
- #define **CC_BITS_IN_BYTE** 8
- #define **CC_BITS_IN_32BIT_WORD** 32
- #define **CC_32BIT_WORD_SIZE** 4
- #define **CC_32BIT_WORD_IN_64BIT_DWORD** 2

2.12.11.3 Enumerations

- enum **CCBool** { **CC_FALSE** = 0, **CC_TRUE** = 1 }

2.12.11.4 Macro Definition Documentation

2.12.11.4.1 #define CALC_32BIT_WORDS_FROM_64BIT_DWORD(sizeWords) (sizeWords *CC_32BIT_WORD_IN_64BIT_DWORD)

This macro calculates the number of full 32-bit words from 64-bits dwords.

2.12.11.4.2 #define CALC_32BIT_WORDS_FROM_BYTES(sizebytes) ((sizebytes)/CC_32BIT_WORD_SIZE + (((sizebytes) & (CC_32BIT_WORD_SIZE-1)) > 0))

This macro calculates the number of full 32-bit words from bytes, where three bytes are one word.

2.12.11.4.3 #define CALC_FULL_32BIT_WORDS(numBits) ((numBits)/CC_BITS_IN_32BIT_WORD + (((numBits) & (CC_BITS_IN_32BIT_WORD-1)) > 0))

This macro calculates the number of full 32-bit words from bits, where 31 bits are one word.

2.12.11.4.4 #define CALC_FULL_BYTES(numBits) ((numBits)/CC_BITS_IN_BYTE + (((numBits) & (CC_BITS_IN_BYTE-1)) > 0))

This macro calculates the number of full bytes from bits, where seven bits are one byte.

2.12.11.4.5 #define CC_1K_SIZE_IN_BYTES 1024

Definition of 1 KB in bytes.

2.12.11.4.6 #define CC_32BIT_WORD_IN_64BIT_DWORD 2

Definition of number of 32-bits words in a 64-bits dword.

2.12.11.4.7 #define CC_32BIT_WORD_SIZE 4

Definition of number of bytes in a 32-bits word.

2.12.11.4.8 #define CC_BITS_IN_32BIT_WORD 32

Definition of number of bits in a 32-bits word.

2.12.11.4.9 #define CC_BITS_IN_BYTE 8

Definition of number of bits in a byte.

2.12.11.4.10 #define CC_FAIL 1UL

Failure definition.

2.12.11.4.11 #define CC_MAX(a, b) max(a , b)

Definition for maximal calculation.

2.12.11.4.12 #define CC_MAX_UINT32_VAL (0xFFFFFFFF)

The maximal uint32 value.

2.12.11.4.13 #define CC_MIN(a, b) min(a , b)

Definition for minimal calculation.

2.12.11.4.14 #define CC_OK 0

Success (OK) definition.

2.12.11.4.15 #define CC_SUCCESS 0UL

Success definition.

2.12.11.4.16 #define CC_UNUSED_PARAM(prm) ((void)prm)

Handles unused parameters in the code, to avoid compilation warnings.

2.12.11.4.17 #define ROUNDUP_BITS_TO_32BIT_WORD(numBits) (CALC_FULL_32BIT_WORDS(numBits) *CC_BITS_IN_32BIT_WORD)

This macro rounds up bits to 32-bit words.

2.12.11.4.18 #define ROUNDUP_BITS_TO_BYTES(numBits) (CALC_FULL_BYTES(numBits) *CC_BITS_IN_BYTE)

This macro rounds up bits to bytes.

2.12.11.4.19 #define ROUNDUP_BYTES_TO_32BIT_WORD(sizebytes) (CALC_32BIT_WORDS_FROM_BYTES (sizebytes) *CC_32BIT_WORD_SIZE)

This macro rounds up bytes to 32-bit words.

2.12.11.5 Enumeration Type Documentation

2.12.11.5.1 enum CCBool

Boolean types.

Enumerator:

Enum	Description
CC_FALSE	Boolean false definition.
CC_TRUE	Boolean true definition.

2.12.12 CryptoCell PAL platform-dependent compiler-related definitions

Contains CryptoCell PAL platform-dependent compiler-related definitions.

2.12.12.1 Files

- file **cc_pal_compiler.h**

  This file contains CryptoCell PAL platform-dependent compiler-related definitions.

2.12.12.2 Macros

- #define **CC_PAL_COMPILER_SECTION**(sectionName) __attribute__((section(sectionName)))
- #define **CC_PAL_COMPILER_KEEP_SYMBOL** __attribute__((used))
- #define **CC_PAL_COMPILER_ALIGN**(alignement) __attribute__((aligned(alignement)))
- #define **CC_PAL_COMPILER_FUNC_NEVER_RETURNS** __attribute__((noreturn))
- #define **CC_PAL_COMPILER_FUNC_DONT_INLINE** __attribute__((noinline))
- #define **CC_PAL_COMPILER_TYPE_MAY_ALIAS** __attribute__((__may_alias__))

- #define **CC_PAL_COMPILER_SIZEOF_STRUCT_MEMBER**(type_name, member_name) sizeof(((type_name *)0)->member_name)
- #define **CC_ASSERT_CONCAT_**(a, b) a##b
- #define **CC_ASSERT_CONCAT**(a, b) **CC_ASSERT_CONCAT_**(a, b)
- #define **CC_PAL_COMPILER_ASSERT**(cond, message) enum { **CC_ASSERT_CONCAT**(assert_line_, __LINE__) = 1/(!!(cond)) }

2.12.12.3 Macro Definition Documentation

2.12.12.3.1 #define CC_ASSERT_CONCAT(a, b) CC_ASSERT_CONCAT_(a, b)

Definition of assertion.

2.12.12.3.2 #define CC_ASSERT_CONCAT_(a, b) a##b

Definition of assertion.

2.12.12.3.3 #define CC_PAL_COMPILER_ALIGN(alignement) __attribute__((aligned(alignement)))

Align a given data item in bytes.

2.12.12.3.4 #define CC_PAL_COMPILER_ASSERT(cond, message) enum { CC_ASSERT_CONCAT(assert_line_, __LINE__) = 1/(!!(cond)) }

Definition of assertion.

2.12.12.3.5 #define CC_PAL_COMPILER_FUNC_DONT_INLINE __attribute__((noinline))

Prevent a function from being inlined.

2.12.12.3.6 #define CC_PAL_COMPILER_FUNC_NEVER_RETURNS __attribute__((noreturn))

Mark a function that never returns.

2.12.12.3.7 #define CC_PAL_COMPILER_KEEP_SYMBOL __attribute__((used))

Mark symbol as used, that is, prevent the garbage collector from dropping it.

2.12.12.3.8 #define CC_PAL_COMPILER_SECTION(sectionName) __attribute__((section(sectionName)))

Associate a symbol with a link section.

2.12.12.3.9 #define CC_PAL_COMPILER_SIZEOF_STRUCT_MEMBER(type_name, member_name) sizeof(((type_name *)0)->member_name)

Get the size of a structure-type member.

2.12.12.3.10 #define CC_PAL_COMPILER_TYPE_MAY_ALIAS __attribute__((__may_alias__))

Given data type might serve as an alias for another data-type pointer.

2.12.13 CryptoCell PAL power-management APIs

Contains PAL power-management APIs.

2.12.13.1 Files

- file **cc_pal_pm.h**

  This file contains the definitions and APIs for power-management implementation.

2.12.13.2 Functions

- void **CC_PalPowerSaveModeInit** (void)

  This function initiates an atomic counter.
- int32_t **CC_PalPowerSaveModeStatus** (void)

  This function returns the number of active registered CryptoCell operations.
- CCError_t **CC_PalPowerSaveModeSelect** (**CCBool** isPowerSaveMode)

  This function updates the atomic counter on each call to CryptoCell.

2.12.13.3 Function Documentation

2.12.13.3.1 void CC_PalPowerSaveModeInit (void)

Returns:

Void.

2.12.13.3.2 CCError_t CC_PalPowerSaveModeSelect (CCBool isPowerSaveMode)

On each call to CryptoCell, the counter is increased. At the end of each operation the counter is decreased. Once the counter is zero, an external callback is called.

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`isPowerSaveMode`	TRUE: CryptoCell is active. FALSE: CryptoCell is idle.

2.12.13.3.3 int32_t CC_PalPowerSaveModeStatus (void)

Returns:

The value of the atomic counter.

2.12.14 CryptoCell PAL TRNG APIs

Contains APIs for retrieving TRNG user parameters.

2.12.14.1 Files

- file **cc_pal_trng.h**

  This file contains APIs for retrieving TRNG user parameters.

2.12.14.2 Data Structures

- struct **CC_PalTrngParams_t**

2.12.14.3 Typedefs

- typedef struct **CC_PalTrngParams_t** **CC_PalTrngParams_t**

2.12.14.4 Functions

- CCError_t **CC_PalTrngParamGet** (**CC_PalTrngParams_t** *pTrngParams, size_t *pParamsSize)

  This function returns the TRNG user parameters.

2.12.14.5 Typedef Documentation

2.12.14.5.1 typedef struct CC_PalTrngParams_t CC_PalTrngParams_t

Definition for the structure of the random-generator parameters of CryptoCell, containing the user-given parameters.

2.12.14.6 Function Documentation

2.12.14.6.1 CCError_t CC_PalTrngParamGet (CC_PalTrngParams_t * pTrngParams, size_t * pParamsSize)

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
out	`pTrngParams`	A pointer to the TRNG user parameters.
in,out	`pParamsSize`	A pointer to the size of the TRNG-user-parameters structure used. Input: the function must verify its size is the same as **CC_PalTrngParams_t**. Output: the function returns the size of **CC_PalTrngParams_t** for library-size verification.

2.12.15 Specific errors of the CryptoCell PAL APIs

Contains platform-dependent PAL-API error definitions.

2.12.15.1 Files

- file **cc_pal_error.h**

  This file contains the error definitions of the platform-dependent PAL APIs.

2.12.15.2 Macros

- #define **CC_PAL_BASE_ERROR** 0x0F000000
- #define **CC_PAL_MEM_BUF1_GREATER CC_PAL_BASE_ERROR** + 0x01UL
- #define **CC_PAL_MEM_BUF2_GREATER CC_PAL_BASE_ERROR** + 0x02UL
- #define **CC_PAL_SEM_CREATE_FAILED CC_PAL_BASE_ERROR** + 0x03UL
- #define **CC_PAL_SEM_DELETE_FAILED CC_PAL_BASE_ERROR** + 0x04UL
- #define **CC_PAL_SEM_WAIT_TIMEOUT CC_PAL_BASE_ERROR** + 0x05UL
- #define **CC_PAL_SEM_WAIT_FAILED CC_PAL_BASE_ERROR** + 0x06UL
- #define **CC_PAL_SEM_RELEASE_FAILED CC_PAL_BASE_ERROR** + 0x07UL
- #define **CC_PAL_ILLEGAL_ADDRESS CC_PAL_BASE_ERROR** + 0x08UL

2.12.15.3 Macro Definition Documentation

2.12.15.3.1 #define CC_PAL_BASE_ERROR 0x0F000000

The PAL error base.

2.12.15.3.2 #define CC_PAL_ILLEGAL_ADDRESS CC_PAL_BASE_ERROR + 0x08UL

Illegal PAL address.

2.12.15.3.3 #define CC_PAL_MEM_BUF1_GREATER CC_PAL_BASE_ERROR + 0x01UL

Buffer one is greater than buffer two error.

2.12.15.3.4 #define CC_PAL_MEM_BUF2_GREATER CC_PAL_BASE_ERROR + 0x02UL

Buffer two is greater than buffer one error.

2.12.15.3.5 #define CC_PAL_SEM_CREATE_FAILED CC_PAL_BASE_ERROR + 0x03UL

Semaphore creation failed.

2.12.15.3.6 #define CC_PAL_SEM_DELETE_FAILED CC_PAL_BASE_ERROR + 0x04UL

Semaphore deletion failed.

2.12.15.3.7 #define CC_PAL_SEM_RELEASE_FAILED CC_PAL_BASE_ERROR + 0x07UL

Semaphore release failed.

2.12.15.3.8 #define CC_PAL_SEM_WAIT_FAILED CC_PAL_BASE_ERROR + 0x06UL

Semaphore wait failed.

2.12.15.3.9 #define CC_PAL_SEM_WAIT_TIMEOUT CC_PAL_BASE_ERROR + 0x05UL

Semaphore reached timeout.

2.13 CryptoCell PKA APIs

Contains all CryptoCell PKA APIs.

2.13.1 Modules

- **CryptoCell PKA-specific definitions**

  Contains the CryptoCell PKA API definitions.

2.13.2 CryptoCell PKA-specific definitions

Contains the CryptoCell PKA API definitions.

2.13.2.1 Modules

- **CryptoCell PKA-API platform-dependent types and definitions**

  Contains the platform-dependent definitions of the CryptoCell PKA APIs.

2.13.2.2 Files

- file **cc_pka_defs_hw.h**

  This file contains all of the enums and definitions that are used in PKA APIs.

2.13.2.3 Macros

- #define **CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS** ((**CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS** + **CC_PKA_WORD_SIZE_IN_BITS**) / **CC_BITS_IN_32BIT_WORD**)
- #define **CC_ECPKI_MODUL_MAX_LENGTH_IN_BITS** 521

- #define **CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS** 5
- #define **CC_PKA_ECPKI_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS** **CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS**
- #define **CC_PKA_BARRETT_MOD_TAG_SIZE_IN_WORDS** (((**CC_PKA_WORD_SIZE_IN_BITS** + **PKA_EXTRA_BITS** - 1) + (**CC_BITS_IN_32BIT_WORD** - 1)) / **CC_BITS_IN_32BIT_WORD**)
- #define **CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS** **CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS**
- #define **CC_PKA_PUB_KEY_BUFF_SIZE_IN_WORDS** (2*__CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS__)
- #define **CC_PKA_PRIV_KEY_BUFF_SIZE_IN_WORDS** (2*__CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS__)
- #define **CC_PKA_KGDATA_BUFF_SIZE_IN_WORDS** (3*__CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS__ + 3*__CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS__)
- #define **CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS** 18
- #define **CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS** (**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS** + 1)
- #define **CC_PKA_DOMAIN_BUFF_SIZE_IN_WORDS** (2*__CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS__)
- #define **COUNT_NAF_WORDS_PER_KEY_WORD** 8
- #define **CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS** (**COUNT_NAF_WORDS_PER_KEY_WORD***__CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS__ + 1)
- #define **CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS** (**CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS**+**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**+2)
- #define **CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS** (3*__CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS__+**CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS**)
- #define **CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS** (6*__CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS__+**CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS**)
- #define **CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS** (2*__CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS__ + **CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS**)
- #define **CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS** (2*__CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS__ + **CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS**)

- #define **CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS** (3***CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**)
- #define **CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_BYTES** 32U
- #define **CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS** 8U
- #define **CC_EC_MONT_TEMP_BUFF_SIZE_IN_32BIT_WORDS** (8 ***CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS**)
- #define **CC_EC_EDW_TEMP_BUFF_SIZE_IN_32BIT_WORDS** (8***CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS** + (sizeof(**CCHashUserContext_t**)+**CC_32BIT_WORD_SIZE**-1)/**CC_32BIT_WORD_SIZE**)

2.13.2.4 Macro Definition Documentation

2.13.2.4.1 #define CC_EC_EDW_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8*CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS + (sizeof(CCHashUserContext_t)+CC_32BIT_WORD_SIZE-1)/CC_32BIT_WORD_SIZE)

The size of the ECC Edwards temporary buffer in words.

2.13.2.4.2 #define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_BYTES 32U

The maximal size of the modulus buffers for CC_EC_MONT and EC_EDW in bytes.

2.13.2.4.3 #define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS 8U

The maximal size of the modulus buffers for CC_EC_MONT and EC_EDW in words.

2.13.2.4.4 #define CC_EC_MONT_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8 *CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS)

The size of the ECC Montgomery temporary buffer in words.

2.13.2.4.5 #define CC_ECPKI_MODUL_MAX_LENGTH_IN_BITS 521

The maximal EC modulus size.

2.13.2.4.6 #define CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS 18

The maximal size of the EC modulus in words.

2.13.2.4.7 #define CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS (CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS + 1)

The maximal size of the EC order in words.

2.13.2.4.8 #define CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS 5

The size of the buffers for Barrett modulus tag NP, used in PKI algorithms.

2.13.2.4.9 #define CC_PKA_BARRETT_MOD_TAG_SIZE_IN_WORDS (((CC_PKA_WORD_SIZE_IN_BITS + PKA_EXTRA_BITS - 1) + (CC_BITS_IN_32BIT_WORD - 1)) / CC_BITS_IN_32BIT_WORD)

The actual size of Barrett modulus tag NP in words for current HW platform.

2.13.2.4.10 #define CC_PKA_DOMAIN_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the EC domain in words.

2.13.2.4.11 #define CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC ECDH temporary-buffer in words.

2.13.2.4.12 #define CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS (COUNT_NAF_WORDS_PER_KEY_WORD*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + 1)

The maximal length of the ECC NAF buffer.

2.13.2.4.13 #define CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS (6*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC sign temporary buffer in words.

2.13.2.4.14 #define CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS)

The size of the ECC verify temporary-buffer in words.

2.13.2.4.15 #define CC_PKA_ECPKI_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS

The size of the buffers for Barrett modulus tag NP, used in ECC.

2.13.2.4.16 #define CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC temporary buffer in words.

2.13.2.4.17 #define CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS (CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS+CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+2)

The size of the Scalar buffer in words.

2.13.2.4.18 #define CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the PKA KG temporary-buffer in words.

2.13.2.4.19 #define CC_PKA_KGDATA_BUFF_SIZE_IN_WORDS (3*CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS + 3*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA KG buffer in words

2.13.2.4.20 #define CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS

The maximal size of the PKA modulus.

2.13.2.4.21 #define CC_PKA_PRIV_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA private-key in words.

2.13.2.4.22 #define CC_PKA_PUB_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA public-key in words.

2.13.2.4.23 #define CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS ((CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS + CC_PKA_WORD_SIZE_IN_BITS) / CC_BITS_IN_32BIT_WORD)

The maximal RSA modulus size.

2.13.2.4.24 #define COUNT_NAF_WORDS_PER_KEY_WORD 8

The ECC NAF buffer definitions.

2.13.3 CryptoCell PKA-API platform-dependent types and definitions

Contains the platform-dependent definitions of the CryptoCell PKA APIs.

2.13.3.1 Files

- file **cc_pka_hw_plat_defs.h**

  This file contains the platform-dependent definitions of the CryptoCell PKA APIs.

2.13.3.2 Macros

- #define **CC_PKA_WORD_SIZE_IN_BITS** 64
- #define **CC_SRP_MAX_MODULUS_SIZE_IN_BITS** 3072
- #define **CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS** 4096
- #define **CC_RSA_MAX_KEY_GENERATION_HW_SIZE_BITS** 3072
- #define **CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_WORDS** **CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS** / **CC_BITS_IN_32BIT_WORD**
- #define **SB_CERT_RSA_KEY_SIZE_IN_BITS** 3072UL
- #define **SB_CERT_RSA_KEY_SIZE_IN_BYTES** (**SB_CERT_RSA_KEY_SIZE_IN_BITS**/**CC_BITS_IN_BYTE**)
- #define **SB_CERT_RSA_KEY_SIZE_IN_WORDS** (**SB_CERT_RSA_KEY_SIZE_IN_BITS**/**CC_BITS_IN_32BIT_WORD**)
- #define **PKA_EXTRA_BITS** 8
- #define **PKA_MAX_COUNT_OF_PHYS_MEM_REGS** 32

2.13.3.3 Macro Definition Documentation

2.13.3.3.1 #define CC_PKA_WORD_SIZE_IN_BITS 64

The size of the PKA engine word.

2.13.3.3.2 #define CC_RSA_MAX_KEY_GENERATION_HW_SIZE_BITS 3072

The maximal supported size of key-generation in RSA in bits.

2.13.3.3.3 #define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS 4096

The maximal supported size of modulus in RSA in bits.

2.13.3.3.4 #define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_WORDS CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS / CC_BITS_IN_32BIT_WORD

The maximal supported size of modulus in RSA in words.

2.13.3.3.5 #define CC_SRP_MAX_MODULUS_SIZE_IN_BITS 3072

The maximal supported size of modulus in bits.

2.13.3.3.6 #define PKA_EXTRA_BITS 8

The maximal count of extra bits in PKA operations.

2.13.3.3.7 #define PKA_MAX_COUNT_OF_PHYS_MEM_REGS 32

The number of memory registers in PKA operations.

2.13.3.3.8 #define SB_CERT_RSA_KEY_SIZE_IN_BITS 3072UL

The size of the RSA public modulus key of the Secure Boot or Secure Debug certificate in bits.

2.13.3.3.9 #define SB_CERT_RSA_KEY_SIZE_IN_BYTES (SB_CERT_RSA_KEY_SIZE_IN_BITS/CC_BITS_IN_BYTE)

The size of the RSA public modulus key of the Secure Boot or Secure Debug certificate in bytes.

2.13.3.3.10 #define SB_CERT_RSA_KEY_SIZE_IN_WORDS (SB_CERT_RSA_KEY_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)

The size of the RSA public modulus key of the Secure Boot or Secure Debug certificate in words.

2.14 CryptoCell RNG APIs

The Random Number Generator (RNG) module supports random number generation, as defined in *NIST SP 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. See **mbedtls_ctr_drbg_random()**.

2.14.1 Modules

- **CryptoCell random-number generation APIs.**

  Contains the CryptoCell random-number generation APIs.

2.14.2 Detailed description

The block-cipher counter-mode based deterministic random-bit generator (CTR_DBRG). CryptoCell provides the source of entropy.

For the implementation of RNG, see **ctr_drbg.h**.

2.14.3 CryptoCell random-number generation APIs

Contains the CryptoCell random-number generation APIs.

2.14.3.1 Files

- file **cc_rnd_common.h**

  This file contains the CryptoCell random-number generation (RNG) APIs.

2.14.3.2 Data Structures

- struct **CCRndWorkBuff_t**
- struct **CCRndState_t**

  The structure for the RND state. This includes internal data that must be saved by the user between boots.

- struct **CCRndContext_t**

2.14.3.3 Macros

- #define **CC_RND_SEED_MAX_SIZE_WORDS** 12
- #define **CC_RND_MAX_GEN_VECTOR_SIZE_BITS** 0x7FFFF
- #define **CC_RND_MAX_GEN_VECTOR_SIZE_BYTES** 0xFFFF
- #define **CC_RND_REQUESTED_SIZE_COUNTER** 0x3FFFF
- #define **CC_RND_WORK_BUFFER_SIZE_WORDS** 136
- #define **CC_RND_TRNG_SRC_INNER_OFFSET_WORDS** 2
- #define **CC_RND_TRNG_SRC_INNER_OFFSET_BYTES** (**CC_RND_TRNG_SRC_INNER_OFFSET_WORDS***sizeof(uint32_t))

2.14.3.4 Typedefs

- typedef int(***CCRndGenerateVectWorkFunc_t**) (void *rndState_ptr, unsigned char *out_ptr, size_t outSizebytes)

2.14.3.5 Enumerations

- enum **CCRndMode_t** { **CC_RND_FE** = 1, **CC_RND_ModeLast** = 0x7FFFFFFF }

2.14.3.6 Functions

- CCError_t **CC_RndSetGenerateVectorFunc** (**CCRndContext_t** *rndContext_ptr, **CCRndGenerateVectWorkFunc_t** rndGenerateVectFunc)

  This function sets the RND vector-generation function into the RND context.

2.14.3.7 Macro Definition Documentation

2.14.3.7.1 #define CC_RND_MAX_GEN_VECTOR_SIZE_BITS 0x7FFFF

The maximal size of the generated vector in bits.

2.14.3.7.2 #define CC_RND_MAX_GEN_VECTOR_SIZE_BYTES 0xFFFF

The maximal size of the generated random vector in bytes.

2.14.3.7.3 #define CC_RND_REQUESTED_SIZE_COUNTER 0x3FFFF

The maximal size of the generated vector in bytes.

2.14.3.7.4 #define CC_RND_SEED_MAX_SIZE_WORDS 12

The maximal size of the random seed in words.

2.14.3.7.5 #define CC_RND_TRNG_SRC_INNER_OFFSET_BYTES (CC_RND_TRNG_SRC_INNER_OFFSET_WORDS*sizeof(uint32_t))

The definition of the internal offset in bytes.

2.14.3.7.6 #define CC_RND_TRNG_SRC_INNER_OFFSET_WORDS 2

The definition of the internal offset in words.

2.14.3.7.7 #define CC_RND_WORK_BUFFER_SIZE_WORDS 136

The size of the temporary buffer in words.

2.14.3.8 Typedef Documentation

2.14.3.8.1 typedef int(*CCRndGenerateVectWorkFunc_t) (void *rndState_ptr, unsigned char *out_ptr, size_t outSizebytes)

The RND vector-generation function pointer.

2.14.3.9 Enumeration Type Documentation

2.14.3.9.1 enum CCRndMode_t

The definition of the random operation modes.

Enumerator:

Enum	Description
`CC_RND_FE`	HW entropy estimation: 800-90B or full.
`CC_RND_ModeLast`	Reserved.

2.14.3.10 Function Documentation

2.14.3.10.1 CCError_t CC_RndSetGenerateVectorFunc (CCRndContext_t * rndContext_ptr, CCRndGenerateVectWorkFunc_t rndGenerateVectFunc)

It is called as part of Arm CryptoCell library initialization, to set the RND vector generation function into the primary RND context.

It must be called before any other API that requires the RND context as a parameter.

Returns:

CC_OK on success.

A non-zero value from cc_rnd_error.h on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer that is allocated by the user, which is used to maintain the RND state, as well as pointers to the functions used for random vector generation.
in	`rndGenerateVectFunc`	A pointer to the CC_RndGenerateVector random vector-generation function.

2.15 CryptoCell RSA APIs

RSA is an asymmetric algorithm used for secure-data transmission.

2.15.1 Modules

- **CryptoCell-312 hardware limitations for RSA**
- **Typical usage of RSA in CryptoCell-312**
- **Typical insertion of keys in CryptoCell-312**

2.15.2 Detailed description

As it is considered slow, it is mainly used to pass encrypted shared keys for symmetric key cryptography.

The RSA module implements the standards defined in *PKCS #1 v1.5 Public-Key Cryptography Standards RSA Encryption Standard* and *PKCS #1 v2.1 Public-Key Cryptography Standards RSA Cryptography Specifications*.

CryptoCell-312 does not support blinding for RSA. If a function receives random pointers as input, these may be NULL.

For the implementation of RSA, see **rsa.h**

2.15.3 CryptoCell-312 hardware limitations for RSA

CryptoCell-312 supports the following RSA key sizes for private-public operations:

- 256 bytes (2048 bits).
- 384 bytes (3072 bits).
- 512 bytes (4096 bits).

For key-generation, CryptoCell-312 supports the following RSA key sizes:

- 256 bytes (2048 bits).
- 384 bytes (3072 bits).

2.15.4 Typical usage of RSA in CryptoCell-312

The following is a typical RSA operation flow:

1. **mbedtls_rsa_init()**.
2. **mbedtls_rsa_gen_key()**.
3. **mbedtls_rsa_pkcs1_encrypt()**.

CryptoCell-312 requires that the same hash_id used for **mbedtls_rsa_init()** is used for all subsequent operations. Otherwise, it returns an error.

2.15.5 Typical insertion of keys in CryptoCell-312

The following is a typical RSA key-insertion flow:

1. **mbedtls_rsa_import()** or **mbedtls_rsa_import_raw()**.
2. **mbedtls_rsa_complete()**.

If you insert keys that are not derived by CryptoCell-312, the following restrictions apply:

- The user may insert N , D , E , and the complete function does not derive the P and Q (the CRT values).
- The user may insert P and Q , and the complete function derives the CRT values from that, but does not derive D.
- It is Illegal to insert only part of the CRT key (only DP for example).
- If all the required key parameters were inserted the function does nothing.

2.16 CryptoCell Secure Boot and Secure Debug APIs

Contains all Secure Boot and Secure Debug APIs and definitions.

2.16.1 Modules

- **CryptoCell Secure Boot and Secure Debug API definitions**

  Contains definitions used for the Secure Boot and Secure Debug APIs.
- **CryptoCell Secure Boot basic type definitions**

  Contains CryptoCell Secure Boot basic type definitions.
- **CryptoCell Secure Boot certificate-chain-processing APIs.**

  Contains CryptoCell Secure Boot certificate-chain-processing APIs.
- **CryptoCell Secure Boot type definitions**

  Contains CryptoCell Secure Boot type definitions.

2.16.2 CryptoCell Secure Boot and Secure Debug API definitions

Contains definitions used for the Secure Boot and Secure Debug APIs.

2.16.2.1 Files

- file **bootimagesverifier_def.h**

  This file contains definitions used for the Secure Boot and Secure Debug APIs.

2.16.2.2 Macros

- #define **CC_SB_MAX_NUM_OF_IMAGES** 16
- #define **CC_SB_MAX_CERT_SIZE_IN_BYTES** (0xB10)
- #define **CC_SB_MAX_CERT_SIZE_IN_WORDS** (**CC_SB_MAX_CERT_SIZE_IN_BYTES**/**CC_32BIT_WORD_SIZE**)
- #define **CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES** (0x350)
- #define **CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES** (**CC_SB_MAX_CERT_SIZE_IN_BYTES** + **CC_MAX**(**CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES**, CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES))

  The minimal size of the Secure Boot workspace in bytes.

2.16.2.3 Macro Definition Documentation

2.16.2.3.1 #define CC_SB_MAX_CERT_SIZE_IN_BYTES (0xB10)

The maximal size of an X.509 certificate in bytes.

2.16.2.3.2 #define CC_SB_MAX_CERT_SIZE_IN_WORDS (CC_SB_MAX_CERT_SIZE_IN_BYTES/CC_32BIT_WORD_SIZE)

The maximal size of a certificate in words.

2.16.2.3.3 #define CC_SB_MAX_NUM_OF_IMAGES 16

The maximal number of SW images per content certificate.

2.16.2.3.4 #define CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES (0x350)

The size of the Secure Debug workspace in bytes. This workspace is used to store RSA parameters. For example, modulus and signature.

2.16.2.3.5 #define CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES (CC_SB_MAX_CERT_SIZE_IN_BYTES + CC_MAX(CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES, CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES))

The Secure Boot APIs use a temporary workspace for processing the data that is read from the flash, before loading the SW modules to their designated memory addresses. This workspace must be large enough to accommodate the size of the certificates, and twice the size of the data that is read from flash in each processing round.

The definition of CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES is comprised of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES and additional space for the certificate itself, which resides in the workspace at the same time the SW images data is processed.

It is assumed that the optimal size of the data to read in each processing round is 4KB, based on the standard flash-memory page size. Therefore, the size of the double buffer, CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES , is defined by default as 8KB in the project configuration file. This can be changed to accommodate the optimal value in different environments. CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES is defined by the Boot Services makefile as equal to CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES.

When writing code that uses the Secure Boot APIs, and includes the **bootimagesverifier_def.h** file, the value of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES must be defined by your makefile to be exactly the same value as was used when compiling the SBROM library. Additionally, CC_SB_X509_CERT_SUPPORTED must be defined in the Makefile, according to the definition of CC_CONFIG_SB_X509_CERT_SUPPORTED.

The size of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES must be a multiple of the hash SHA-256 block size of 64 bytes.

2.16.3 CryptoCell Secure Boot basic type definitions

Contains CryptoCell Secure Boot basic type definitions.

2.16.4 CryptoCell Secure Boot certificate-chain-processing APIs

Contains CryptoCell Secure Boot certificate-chain-processing APIs.

2.16.4.1 Files

- file **mbedtls_cc_sbrt.h**

  This file contains CryptoCell Secure Boot certificate-chain processing APIs.

2.16.4.2 Functions

- CCError_t **mbedtls_sb_cert_chain_cerification_init** (**CCSbCertInfo_t** *certPkgInfo)

  This function initializes the Secure Boot certificate-chain processing.

- CCError_t **mbedtls_sb_cert_verify_single** (**CCSbFlashReadFunc** flashReadFunc, void *userContext, **CCAddr_t** certStoreAddress, **CCSbCertInfo_t** *pCertPkgInfo, uint32_t *pHeader, uint32_t headerSize, uint32_t *pWorkspace, uint32_t workspaceSize)

  This function verifies a single certificate package containing either a key or content certificate.

- CCError_t **mbedtls_sb_sw_image_store_address_change** (uint32_t *pCert, uint32_t maxCertSizeWords, **CCAddr_t** address, uint32_t indexOfAddress)

  This function changes the storage address of a specific SW image in the content certificate.

2.16.4.3 Function Documentation

2.16.4.3.1 CCError_t mbedtls_sb_cert_chain_cerification_init (CCSbCertInfo_t * certPkgInfo)

It initializes the internal data fields of the certificate package.

This function must be the first API called when processing a Secure Boot certificate chain.

Returns:

CC_OK on success.

A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in,out	`certPkgInfo`	A pointer to the information about the certificate package.

2.16.4.3.2 CCError_t mbedtls_sb_cert_verify_single (CCSbFlashReadFunc flashReadFunc, void * userContext, CCAddr_t certStoreAddress, CCSbCertInfo_t * pCertPkgInfo, uint32_t * pHeader, uint32_t headerSize, uint32_t * pWorkspace, uint32_t workspaceSize)

It verifies the following:

- The public key as saved in the certificate, against its hash. Its hash is found in either the OTP memory (HBK) or in certPkgInfo.
- The RSA signature of the certificate.
- The SW version in the certificate is higher than or equal to the minimal SW version, as recorded on the device and passed in certPkgInfo.
- For content certificates: Each SW module against its hash in the certificate.

The certificates may reside in the memory or in the flash. The flashReadFunc() must be implemented accordingly.

The certificates and images must both be placed either in the memory or in the flash.

Returns:

CC_OK on success.

A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`flashReadFunc`	A pointer to the flash-read function.
in	`userContext`	An additional pointer for flashRead() usage. May be NULL.
in	`certStoreAddress`	The address where the certificate is located. This address is provided to flashReadFunc.
in,out	`pCertPkgInfo`	A pointer to the certificate-package information.
in,out	`pHeader`	A pointer to a buffer used for extracting the X.509 TBS Headers. Note: Must be NULL for proprietary certificates.
in	`headerSize`	The size of pHeader in bytes. Note: Must be 0 for proprietary certificates.
in	`pWorkspace`	A buffer for the internal use of the function.
in	`workspaceSize`	The size of the workspace in bytes. Note: Must be at least **CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES**.

2.16.4.3.3 CCError_t mbedtls_sb_sw_image_store_address_change (uint32_t * pCert, uint32_t maxCertSizeWords, CCAddr_t address, uint32_t indexOfAddress)

The certificate must be loaded to the RAM before calling this function.

The function does not verify the certificate before the address change.

Returns:

CC_OK on success.

A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`pCert`	The certificate address after it has been loaded to memory.
in	`maxCertSizeWords`	The maximal memory size allocated for the certificate in words (certificate boundaries).
in	`address`	The new storage address to change to.
in	`indexOfAddress`	The index of the SW image in the content certificate, starting from 0.

2.16.5 CryptoCell Secure Boot type definitions

Contains CryptoCell Secure Boot type definitions.

2.16.5.1 Modules

- **CryptoCell Secure Boot and Secure Debug definitions and structures**

  Contains CryptoCell Secure Boot and Secure Debug definitions and structures.

2.16.5.2 Files

- file **secureboot_defs.h**

  This file contains type definitions for the Secure Boot.

2.16.5.3 Data Structures

- struct **CCSbCertInfo_t**

2.16.5.4 Macros

- #define **SW_REC_SIGNED_DATA_SIZE_IN_BYTES** 44
- #define **SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES** 8
- #define **SW_REC_NONE_SIGNED_DATA_SIZE_IN_WORDS** **SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES**/**CC_32BIT_WORD_SIZE**
- #define **CC_SW_COMP_NO_MEM_LOAD_INDICATION** 0xFFFFFFFFUL

2.16.5.5 Macro Definition Documentation

2.16.5.5.1 #define CC_SW_COMP_NO_MEM_LOAD_INDICATION 0xFFFFFFFFUL

Indication whether or not to load the SW image to memory.

2.16.5.5.2 #define SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES 8

The size of the additional-data of the SW-image certificate in bytes.

2.16.5.5.3 #define SW_REC_NONE_SIGNED_DATA_SIZE_IN_WORDS SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES/CC_32BIT_WORD_SIZE

The size of the additional-data of the SW-image certificate in words.

2.16.5.5.4 #define SW_REC_SIGNED_DATA_SIZE_IN_BYTES 44

The size of the data of the SW-image certificate.

2.16.6 CryptoCell Secure Boot and Secure Debug definitions and structures

Contains CryptoCell Secure Boot and Secure Debug definitions and structures.

2.16.6.1 Files

- file **secureboot_gen_defs.h**

  This file contains all of the definitions and structures used for the Secure Boot and Secure Debug.

2.16.6.2 Macros

- #define **CC_SB_MAX_SIZE_ADDITIONAL_DATA_BYTES** 128

2.16.6.3 Typedefs

- typedef uint32_t **CCSbCertPubKeyHash_t**[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t **CCSbCertSocId_t**[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t(***CCSbFlashReadFunc**) (**CCAddr_t** flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

  Typedef of the pointer to the Flash read function that you must implement.
- typedef uint32_t(***CCBsvFlashWriteFunc**) (**CCAddr_t** flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

  Typedef of the pointer to the Flash write function that you must implement.

2.16.6.4 Macro Definition Documentation

2.16.6.4.1 #define CC_SB_MAX_SIZE_ADDITIONAL_DATA_BYTES 128

The maximal size of the additional-data of the Secure Boot in bytes.

2.16.6.5 Typedef Documentation

2.16.6.5.1 typedef uint32_t(*CCBsvFlashWriteFunc) (CCAddr_t flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

The Flash write function is called to write authenticated and decrypted SW modules to flash memory.

2.16.6.5.2 typedef uint32_t CCSbCertPubKeyHash_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of public key hash array.

2.16.6.5.3 typedef uint32_t CCSbCertSocId_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of SoC_ID array.

2.16.6.5.4 typedef uint32_t(*CCSbFlashReadFunc) (CCAddr_t flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

The Flash read function is called to read the certificates and SW modules from flash memory.

It is your responsibility to verify that this function does not copy data from restricted memory regions.

2.17 CryptoCell SRAM mapping APIs

Contains internal SRAM mapping APIs.

2.17.1 Files

- file **cc_sram_map.h**

  This file contains internal SRAM mapping definitions.

2.17.2 Macros

- #define **CC_SRAM_PKA_BASE_ADDRESS** 0x0
- #define **CC_PKA_SRAM_SIZE_IN_KBYTES** 6
- #define **CC_SRAM_RND_HW_DMA_ADDRESS** 0x0
- #define **CC_SRAM_RND_MAX_SIZE** 0x800
- #define **CC_SRAM_MAX_SIZE** 0x1000

2.17.3 Macro definition documentation

2.17.3.1 #define CC_PKA_SRAM_SIZE_IN_KBYTES 6

The size of the PKA SRAM in KB.

2.17.3.2 #define CC_SRAM_MAX_SIZE 0x1000

The maximal size of SRAM.

2.17.3.3 #define CC_SRAM_PKA_BASE_ADDRESS 0x0

The base address of the PKA in the PKA SRAM.

2.17.3.4 #define CC_SRAM_RND_HW_DMA_ADDRESS 0x0

The SRAM address of the RND.

2.17.3.5 #define CC_SRAM_RND_MAX_SIZE 0x800

Addresses 0K-2KB in SRAM. Reserved for RND operations.

2.18 CryptoCell SRP APIs

Contains CryptoCell SRP APIs.

2.18.1 Modules

- **Specific errors of the CryptoCell SRP APIs**

  Contains the CryptoCell SRP-API error definitions.

2.18.2 Files

- file **mbedtls_cc_srp.h**

  This file contains all of the CryptoCell SRP APIs, their enums and definitions.

2.18.3 Data structures

- struct **mbedtls_srp_group_param**

  Group parameters for the SRP.

- struct **mbedtls_srp_context**

2.18.4 Macros

- #define **CC_SRP_MODULUS_SIZE_1024_BITS** 1024
- #define **CC_SRP_MODULUS_SIZE_1536_BITS** 1536
- #define **CC_SRP_MODULUS_SIZE_2048_BITS** 2048
- #define **CC_SRP_MODULUS_SIZE_3072_BITS** 3072
- #define **CC_SRP_MAX_MODULUS_IN_BITS CC_SRP_MODULUS_SIZE_3072_BITS**
- #define **CC_SRP_MAX_MODULUS** (**CC_SRP_MAX_MODULUS_IN_BITS**/**CC_BITS_IN_BYTE**)
- #define **CC_SRP_MAX_MODULUS_IN_WORDS** (**CC_SRP_MAX_MODULUS_IN_BITS**/**CC_BITS_IN_32BIT_WORD**)
- #define **CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS** (256)
- #define **CC_SRP_PRIV_NUM_MIN_SIZE** (**CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS**/**CC_BITS_IN_BYTE**)
- #define **CC_SRP_PRIV_NUM_MIN_SIZE_IN_WORDS** (**CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS**/**CC_BITS_IN_32BIT_WORD**)
- #define **CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS** (**CC_SRP_MAX_MODULUS_IN_BITS**)
- #define **CC_SRP_PRIV_NUM_MAX_SIZE** (**CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS**/**CC_BITS_IN_BYTE**)
- #define **CC_SRP_PRIV_NUM_MAX_SIZE_IN_WORDS** (**CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS**/**CC_BITS_IN_32BIT_WORD**)
- #define **CC_SRP_MAX_DIGEST_IN_WORDS CC_HASH_RESULT_SIZE_IN_WORDS**

- #define **CC_SRP_MAX_DIGEST** (**CC_SRP_MAX_DIGEST_IN_WORDS***CC_32BIT_WORD_SIZE**)
- #define **CC_SRP_MIN_SALT_SIZE** (8)
- #define **CC_SRP_MIN_SALT_SIZE_IN_WORDS** (**CC_SRP_MIN_SALT_SIZE**/**CC_32BIT_WORD_SIZE**)
- #define **CC_SRP_MAX_SALT_SIZE** (64)
- #define **CC_SRP_MAX_SALT_SIZE_IN_WORDS** (**CC_SRP_MAX_SALT_SIZE**/**CC_32BIT_WORD_SIZE**)
- #define **CC_SRP_HK_INIT**(srpType, srpModulus, srpGen, modSizeInBits, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx) **mbedtls_srp_init**(srpType, **CC_SRP_VER_HK**, srpModulus, srpGen, modSizeInBits, **CC_HASH_SHA512_mode**, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx)

2.18.5 typedefs

- typedef uint8_t **mbedtls_srp_modulus**[**CC_SRP_MAX_MODULUS**]
- typedef uint8_t **mbedtls_srp_digest**[**CC_SRP_MAX_DIGEST**]
- typedef uint8_t **mbedtls_srp_sessionKey**[2 ***CC_SRP_MAX_DIGEST**]
- typedef struct **mbedtls_srp_group_param mbedtls_srp_group_param**

  Group parameters for the SRP.
- typedef struct **mbedtls_srp_context mbedtls_srp_context**

2.18.6 Enumerations

- enum **mbedtls_srp_version_t** { **CC_SRP_VER_3** = 0, **CC_SRP_VER_6** = 1, **CC_SRP_VER_6A** = 2, **CC_SRP_VER_HK** = 3, **CC_SRP_NumOfVersions**, **CC_SRP_VersionLast** = 0x7FFFFFFF }
- enum **mbedtls_srp_entity_t** { **CC_SRP_HOST** = 1, **CC_SRP_USER** = 2, **CC_SRP_NumOfEntityType**, **CC_SRP_EntityLast** = 0x7FFFFFFF }

2.18.7 Functions

- CIMPORT_C CCError_t **mbedtls_srp_init** (**mbedtls_srp_entity_t** srpType, **mbedtls_srp_version_t** srpVer, **mbedtls_srp_modulus** srpModulus, uint8_t srpGen, size_t modSizeInBits, **CCHashOperationMode_t** hashMode, uint8_t *pUserName, size_t userNameSize, uint8_t *pPwd, size_t pwdSize, **CCRndContext_t** *pRndCtx, **mbedtls_srp_context** *pCtx)

  This function initiates the SRP context.
- CIMPORT_C CCError_t **mbedtls_srp_pwd_ver_create** (size_t saltSize, uint8_t *pSalt, **mbedtls_srp_modulus** pwdVerifier, **mbedtls_srp_context** *pCtx)

  This function calculates pSalt and pwdVerifier.

- CIMPORT_C CCError_t **mbedtls_srp_clear** (**mbedtls_srp_context** *pCtx)

  This function clears the SRP context.

- CIMPORT_C CCError_t **mbedtls_srp_host_pub_key_create** (size_t ephemPrivSize, **mbedtls_srp_modulus** pwdVerifier, **mbedtls_srp_modulus** hostPubKeyB, **mbedtls_srp_context** *pCtx)

  This function generates the public and private host ephemeral keys, known as B and b in *RFC-5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication*.

- CIMPORT_C CCError_t **mbedtls_srp_host_proof_verify_and_calc** (size_t saltSize, uint8_t *pSalt, **mbedtls_srp_modulus** pwdVerifier, **mbedtls_srp_modulus** userPubKeyA, **mbedtls_srp_modulus** hostPubKeyB, **mbedtls_srp_digest** userProof, **mbedtls_srp_digest** hostProof, **mbedtls_srp_sessionKey** sessionKey, **mbedtls_srp_context** *pCtx)

  This function verifies the user proof, and calculates the host-message proof.

- CIMPORT_C CCError_t **mbedtls_srp_user_pub_key_create** (size_t ephemPrivSize, **mbedtls_srp_modulus** userPubKeyA, **mbedtls_srp_context** *pCtx)

  This function generates public and private user ephemeral keys, known as A and a in *RFC-5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication*.

- CIMPORT_C CCError_t **mbedtls_srp_user_proof_calc** (size_t saltSize, uint8_t *pSalt, **mbedtls_srp_modulus** userPubKeyA, **mbedtls_srp_modulus** hostPubKeyB, **mbedtls_srp_digest** userProof, **mbedtls_srp_sessionKey** sessionKey, **mbedtls_srp_context** *pCtx)

  This function calculates the user proof.

- CIMPORT_C CCError_t **mbedtls_srp_user_proof_verify** (**mbedtls_srp_sessionKey** sessionKey, **mbedtls_srp_modulus** userPubKeyA, **mbedtls_srp_digest** userProof, **mbedtls_srp_digest** hostProof, **mbedtls_srp_context** *pCtx)

  This function verifies the host proof.

2.18.8 Macro definition documentation

2.18.8.1 #define CC_SRP_HK_INIT(srpType, srpModulus, srpGen, modSizeInBits, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx) mbedtls_srp_init(srpType, CC_SRP_VER_HK, srpModulus, srpGen, modSizeInBits, CC_HASH_SHA512_mode, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx)

Macro definition for a specific SRP-initialization function.

2.18.8.2 #define CC_SRP_MAX_DIGEST (CC_SRP_MAX_DIGEST_IN_WORDS*CC_32BIT_WORD_SIZE)

The maximal size of the SRP hash digest in bytes.

2.18.8.3 #define CC_SRP_MAX_DIGEST_IN_WORDS CC_HASH_RESULT_SIZE_IN_WORDS

The maximal size of the SRP hash digest in words.

2.18.8.4 #define CC_SRP_MAX_MODULUS (CC_SRP_MAX_MODULUS_IN_BITS/CC_BITS_IN_BYTE)

The maximal size of the SRP modulus in bytes.

2.18.8.5 #define CC_SRP_MAX_MODULUS_IN_BITS CC_SRP_MODULUS_SIZE_3072_BITS

The maximal size of the SRP modulus in bits.

2.18.8.6 #define CC_SRP_MAX_MODULUS_IN_WORDS (CC_SRP_MAX_MODULUS_IN_BITS/CC_BITS_IN_32BIT_WORD)

The maximal size of the SRP modulus in words.

2.18.8.7 #define CC_SRP_MAX_SALT_SIZE (64)

The maximal size of the salt in bytes.

2.18.8.8 #define CC_SRP_MAX_SALT_SIZE_IN_WORDS (CC_SRP_MAX_SALT_SIZE/CC_32BIT_WORD_SIZE)

The maximal size of the salt in words.

2.18.8.9 #define CC_SRP_MIN_SALT_SIZE (8)

The minimal size of the salt in bytes.

2.18.8.10 #define CC_SRP_MIN_SALT_SIZE_IN_WORDS (CC_SRP_MIN_SALT_SIZE/CC_32BIT_WORD_SIZE)

The minimal size of the salt in words.

2.18.8.11 #define CC_SRP_MODULUS_SIZE_1024_BITS 1024

SRP modulus size of 1024 bits.

2.18.8.12 #define CC_SRP_MODULUS_SIZE_1536_BITS 1536

SRP modulus size of 1536 bits.

2.18.8.13 #define CC_SRP_MODULUS_SIZE_2048_BITS 2048

SRP modulus size of 2048 bits.

2.18.8.14 #define CC_SRP_MODULUS_SIZE_3072_BITS 3072

SRP modulus size of 3072 bits.

2.18.8.15 #define CC_SRP_PRIV_NUM_MAX_SIZE (CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS/CC_BITS_IN_BYTE)

The maximal size of the SRP private number in bytes.

2.18.8.16 #define CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS (CC_SRP_MAX_MODULUS_IN_BITS)

The maximal size of the SRP private number in bits.

2.18.8.17 #define CC_SRP_PRIV_NUM_MAX_SIZE_IN_WORDS (CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)

The maximal size of the SRP private number in words.

2.18.8.18 #define CC_SRP_PRIV_NUM_MIN_SIZE (CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS/CC_BITS_IN_BYTE)

The minimal size of the SRP private number in bytes.

2.18.8.19 #define CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS (256)

The minimal size of the SRP private number in bits.

2.18.8.20 #define CC_SRP_PRIV_NUM_MIN_SIZE_IN_WORDS (CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)

The minimal size of the SRP private number in words.

2.18.9 typedef documentation

2.18.9.1 typedef struct mbedtls_srp_context mbedtls_srp_context

The SRP context prototype

2.18.9.2 typedef uint8_t mbedtls_srp_digest[CC_SRP_MAX_DIGEST]

The definition of the SRP digest buffer.

2.18.9.3 typedef struct mbedtls_srp_group_param mbedtls_srp_group_param

Defines the modulus and the generator used.

2.18.9.4 typedef uint8_t mbedtls_srp_modulus[CC_SRP_MAX_MODULUS]

The definition of the SRP modulus buffer.

2.18.9.5 typedef uint8_t mbedtls_srp_sessionKey[2 *CC_SRP_MAX_DIGEST]

The definition of the SRP session key.

2.18.10 Enumeration type documentation

2.18.10.1 enum mbedtls_srp_entity_t

SRP entity types.

Enumerator:

Enum	Description
`CC_SRP_HOST`	The host entity, also known as server, verifier, or accessory.
`CC_SRP_USER`	The user entity, also known as client, or device.
`CC_SRP_NumOfEntityType`	The maximal number of entities types.
`CC_SRP_EntityLast`	Reserved.

2.18.10.2 enum mbedtls_srp_version_t

Supported SRP versions.

Enumerator:

Enum	Description
`CC_SRP_VER_3`	SRP version 3.
`CC_SRP_VER_6`	SRP version 6.
`CC_SRP_VER_6A`	SRP version 6A.
`CC_SRP_VER_HK`	SRP version HK.
`CC_SRP_NumOfVersions`	The maximal number of supported versions.
`CC_SRP_VersionLast`	Reserved.

2.18.11 Function documentation

2.18.11.1 CIMPORT_C CCError_t mbedtls_srp_clear (mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h**.

Parameters:

I/O	Parameter	Description
in,out	`pCtx`	A pointer to the SRP context.

2.18.11.2 CIMPORT_C CCError_t mbedtls_srp_host_proof_verify_and_calc (size_t saltSize, uint8_t * pSalt, mbedtls_srp_modulus pwdVerifier, mbedtls_srp_modulus userPubKeyA, mbedtls_srp_modulus hostPubKeyB, mbedtls_srp_digest userProof, mbedtls_srp_digest hostProof, mbedtls_srp_sessionKey sessionKey, mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h**.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt. The range is between **CC_SRP_MIN_SALT_SIZE** and **CC_SRP_MAX_SALT_SIZE**.
in	`pSalt`	A pointer to the pSalt number.
in	`pwdVerifier`	A pointer to the password verifier (v).
in	`userPubKeyA`	A pointer to the ephemeral public key of the user (A).
in	`hostPubKeyB`	A pointer to the ephemeral public key of the host (B).
in	`userProof`	A pointer to the SRP user-proof buffer (M1).
out	`hostProof`	A pointer to the SRP host-proof buffer (M2).
out	`sessionKey`	A pointer to the SRP session key (K).
in	`pCtx`	A pointer to the SRP context.

2.18.11.3 CIMPORT_C CCError_t mbedtls_srp_host_pub_key_create (size_t ephemPrivSize, mbedtls_srp_modulus pwdVerifier, mbedtls_srp_modulus hostPubKeyB, mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h** or cc_rnd_error.h.

Parameters:

I/O	Parameter	Description
in	`ephemPrivSize`	The size of the generated ephemeral private key (b). The range is between **CC_SRP_PRIV_NUM_MIN_SIZE** and **CC_SRP_PRIV_NUM_MAX_SIZE**
in	`pwdVerifier`	A pointer to the verifier (v).
out	`hostPubKeyB`	A pointer to the host ephemeral public key (B).
in,out	`pCtx`	A pointer to the SRP context.

2.18.11.4 CIMPORT_C CCError_t mbedtls_srp_init (mbedtls_srp_entity_t srpType, mbedtls_srp_version_t srpVer, mbedtls_srp_modulus srpModulus, uint8_t srpGen, size_t modSizeInBits, CCHashOperationMode_t hashMode, uint8_t * pUserName, size_t userNameSize, uint8_t * pPwd, size_t pwdSize, CCRndContext_t * pRndCtx, mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure as defined in **mbedtls_cc_srp_error.h**.

Parameters:

I/O	Parameter	Description
in	`srpType`	The SRP entity type.
in	`srpVer`	The SRP version.
in	`srpModulus`	A pointer to the SRP modulus, BE byte buffer.
in	`srpGen`	The SRP generator param.
in	`modSizeInBits`	The size of the SRP modulus in bits. Valid values are: 1024 bits, 1536 bits, 2048 bits, or 3072 bits.
in	`hashMode`	The hash mode.
in	`pUserName`	A pointer to the username.
in	`userNameSize`	The size of the username buffer. Must be larger than 0.

I/O	Parameter	Description
in	`pPwd`	A pointer to the user password.
in	`pwdSize`	The size of the user-password buffer. Must be larger than 0 if pPwd is valid.
in	`pRndCtx`	A pointer to the RND context.
out	`pCtx`	A pointer to the SRP host context.

2.18.11.5 CIMPORT_C CCError_t mbedtls_srp_pwd_ver_create (size_t saltSize, uint8_t * pSalt, mbedtls_srp_modulus pwdVerifier, mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h**, cc_rnd_error.h.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt to generate. The range is between **CC_SRP_MIN_SALT_SIZE** and **CC_SRP_MAX_SALT_SIZE**.
out	`pSalt`	A pointer to the pSalt number (s).
out	`pwdVerifier`	A pointer to the password verifier (v).
out	`pCtx`	A pointer to the SRP context.

2.18.11.6 CIMPORT_C CCError_t mbedtls_srp_user_proof_calc (size_t saltSize, uint8_t * pSalt, mbedtls_srp_modulus userPubKeyA, mbedtls_srp_modulus hostPubKeyB, mbedtls_srp_digest userProof, mbedtls_srp_sessionKey sessionKey, mbedtls_srp_context * pCtx)

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h**.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt. The range is between **CC_SRP_MIN_SALT_SIZE** and **CC_SRP_MAX_SALT_SIZE**.
in	`pSalt`	A pointer to the pSalt number.
in	`userPubKeyA`	A pointer to the public ephmeral key of the user (A).
in	`hostPubKeyB`	A pointer to the public ephmeral key of the host (B).
out	`userProof`	A pointer to the SRP user proof buffer (M1).
out	`sessionKey`	A pointer to the SRP session key (K).
out	`pCtx`	A pointer to the SRP context.

2.18.11.7 CIMPORT_C CCError_t mbedtls_srp_user_proof_verify (mbedtls_srp_sessionKey **sessionKey,** mbedtls_srp_modulus **userPubKeyA,** mbedtls_srp_digest **userProof,** mbedtls_srp_digest **hostProof,** mbedtls_srp_context * **pCtx)**

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h**.

Parameters:

I/O	Parameter	Description
in	`sessionKey`	A pointer to the SRP session key (K).
in	`userPubKeyA`	A pointer to the public ephmeral key of the user (A).
in	`userProof`	A pointer to the SRP user proof buffer (M1).
in	`hostProof`	A pointer to the SRP host proof buffer (M2).
out	`pCtx`	A pointer to the SRP user context.

2.18.11.8 CIMPORT_C CCError_t mbedtls_srp_user_pub_key_create (size_t **ephemPrivSize,** mbedtls_srp_modulus **userPubKeyA,** mbedtls_srp_context * **pCtx)**

Returns:

CC_OK on success.

A non-zero value on failure, as defined in **mbedtls_cc_srp_error.h** or cc_rnd_error.h.

Parameters:

I/O	Parameter	Description
in	`ephemPrivSize`	The size of the generated ephemeral private key (a). The range is between **CC_SRP_PRIV_NUM_MIN_SIZE** and **CC_SRP_PRIV_NUM_MAX_SIZE**. The size must be 32 bit aligned
out	`userPubKeyA`	A pointer to the user ephemeral public key (A).
in,out	`pCtx`	A pointer to the SRP context.

2.18.12 Specific errors of the CryptoCell SRP APIs

Contains the CryptoCell SRP-API error definitions.

2.18.12.1 Files

- file **mbedtls_cc_srp_error.h**

  This file contains the error definitions of the CryptoCell SRP APIs.

2.18.12.2 Macros

- #define **CC_SRP_PARAM_INVALID_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x01UL)
- #define **CC_SRP_MOD_SIZE_INVALID_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x02UL)
- #define **CC_SRP_STATE_UNINITIALIZED_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x03UL)
- #define **CC_SRP_RESULT_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x04UL)
- #define **CC_SRP_PARAM_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x05UL)
- #define **CC_SRP_INTERNAL_ERROR** (**CC_SRP_MODULE_ERROR_BASE** + 0x06UL)

2.18.12.3 Macro Definition Documentation

2.18.12.3.1 #define CC_SRP_INTERNAL_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x06UL)

Internal PKI error.

2.18.12.3.2 #define CC_SRP_MOD_SIZE_INVALID_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x02UL)

Illegal modulus size.

2.18.12.3.3 #define CC_SRP_PARAM_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x05UL)

Invalid parameter.

2.18.12.3.4 #define CC_SRP_PARAM_INVALID_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x01UL)

Illegal parameter.

2.18.12.3.5 #define CC_SRP_RESULT_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x04UL)

Result validation error.

2.18.12.3.6 #define CC_SRP_STATE_UNINITIALIZED_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x03UL)

Illegal state (uninitialized).

2.19 CryptoCell utility APIs

This contains all utility APIs.

2.19.1 Modules

- **CryptoCell runtime-library asset-provisioning APIs**

  Contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

- **CryptoCell utility APIs general definitions**

  Contains CryptoCell utility APIs general definitions.

- **CryptoCell utility key-derivation APIs**

  Contains the CryptoCell utility key-derivation function API.

- **Specific errors of the CryptoCell utility module APIs**

  Contains utility API error definitions.

2.19.2 CryptoCell runtime-library asset-provisioning APIs

Contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

2.19.2.1 Files

- file **mbedtls_cc_util_asset_prov.h**

This file contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

2.19.2.2 Macros

- #define **CC_ASSET_PROV_MAX_ASSET_PKG_SIZE** 4144

2.19.2.3 Enumerations

- enum **CCAssetProvKeyType_t** { **ASSET_PROV_KEY_TYPE_KPICV** = 1, **ASSET_PROV_KEY_TYPE_KCP** = 2, **ASSET_PROV_KEY_TYPE_RESERVED** = 0x7FFFFFFF }

2.19.2.4 Functions

- CCError_t **mbedtls_util_asset_pkg_unpack** (**CCAssetProvKeyType_t** keyType, uint32_t assetId, uint32_t *pAssetPackage, size_t assetPackageLen, uint32_t *pAssetData, size_t *pAssetDataLen)

  This function securely provisions ICV or OEM assets to devices using CryptoCell.

2.19.2.5 Macro definition documentation

2.19.2.5.1 #define CC_ASSET_PROV_MAX_ASSET_PKG_SIZE 4144

The maximal size of an asset package.

2.19.2.6 Enumeration type documentation

2.19.2.6.1 enum CCAssetProvKeyType_t

The type of key used to pack the asset.

Enumerator:

Enum	Description
`ASSET_PROV_KEY_TYPE_KPICV`	The ICV provisioning key (Kpicv) key was used to pack the asset.
`ASSET_PROV_KEY_TYPE_KCP`	The OEM provisioning key (Kcp) key was used to pack the asset.
`ASSET_PROV_KEY_TYPE_RESERVED`	Reserved.

2.19.2.7 Function documentation

2.19.2.7.1 CCError_t mbedtls_util_asset_pkg_unpack (CCAssetProvKeyType_t **keyType**, uint32_t **assetId**, uint32_t * **pAssetPackage**, size_t **assetPackageLen**, uint32_t * **pAssetData**, size_t * **pAssetDataLen**)

The function:

1. Receives an encrypted and autenticated asset package.

   This asset package is produced by the ICV or OEM asset-packaging offline utility (using AES-CCM with key derived from Kpicv or Kcp respectively, and the asset identifier).
2. Authenticates the asset package.
3. Decrypts the asset package.
4. Returns the decrypted asset data to the caller.

The function is valid in all life-cycle states. However, an error is returned if the requested key is locked.

Returns:

CC_UTIL_OK on success.

A non-zero value on failure, as defined in **cc_util_error.h**.

Parameters:

I/O	Parameter	Description
in	`keyType`	The type of key used to pack the asset.
in	`assetId`	A 32-bit index identifying the asset, in big-endian order.
in	`pAssetPackage`	The encrypted and authenticated asset package.
in	`assetPackageLen`	The length of the asset package. Must not exceed CC_ASSET_PROV_MAX_ASSET_PKG_SIZE.
out	`pAssetData`	The buffer for retrieving the decrypted asset data.
in,out	`pAssetDataLen`	In: The size of the available asset-data buffer. Maximal size is 4KB. Out: A pointer to the actual length of the decrypted asset data.

2.19.3 CryptoCell utility APIs general definitions

Contains CryptoCell utility APIs general definitions.

2.19.3.1 Files

- file **mbedtls_cc_util_defs.h**

  This file contains general definitions of the CryptoCell utility APIs.

2.19.3.2 Data structures

- struct **mbedtls_util_keydata**

2.19.3.3 Macros

- #define **CC_UTIL_AES_128BIT_SIZE** 16
- #define **CC_UTIL_AES_192BIT_SIZE** 24
- #define **CC_UTIL_AES_256BIT_SIZE** 32
- #define **CC_UTIL_CMAC_DERV_MIN_DATA_IN_SIZE** **CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES**+2
- #define **CC_UTIL_CMAC_DERV_MAX_DATA_IN_SIZE** **CC_UTIL_MAX_KDF_SIZE_IN_BYTES**
- #define **CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES** 0x10UL
- #define **CC_UTIL_AES_CMAC_RESULT_SIZE_IN_WORDS** (**CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES**/sizeof(uint32_t))

2.19.3.4 typedefs

- typedef uint32_t **CCUtilError_t**
- typedef struct **mbedtls_util_keydata** **mbedtls_util_keydata**

2.19.3.5 Macro definition documentation

2.19.3.5.1 #define CC_UTIL_AES_128BIT_SIZE 16

The size of the AES 128-bit key in bytes.

2.19.3.5.2 #define CC_UTIL_AES_192BIT_SIZE 24

The size of the AES 192-bit key in bytes.

2.19.3.5.3 #define CC_UTIL_AES_256BIT_SIZE 32

The size of the AES 256-bit key in bytes.

2.19.3.5.4 #define CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES 0x10UL

The size of the AES CMAC result in bytes.

2.19.3.5.5 #define CC_UTIL_AES_CMAC_RESULT_SIZE_IN_WORDS (CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES/sizeof(uint32_t))

The size of the AES CMAC result in words.

2.19.3.5.6 #define CC_UTIL_CMAC_DERV_MAX_DATA_IN_SIZE CC_UTIL_MAX_KDF_SIZE_IN_BYTES

The maximal size of the data for CMAC derivation operation.

2.19.3.5.7 #define CC_UTIL_CMAC_DERV_MIN_DATA_IN_SIZE CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES +2

The minimal size of the data for the CMAC derivation operation.

2.19.3.6 typedef documentation

2.19.3.6.1 typedef uint32_t CCUtilError_t

Util error type.

2.19.3.6.2 typedef struct mbedtls_util_keydata mbedtls_util_keydata

The key data.

2.19.4 CryptoCell utility general key definitions

Contains KDF API definitions.

2.19.4.1 Files

- file **mbedtls_cc_util_key_derivation_defs.h**

  This file contains the definitions for the key-derivation API.

2.19.4.2 Macros

- #define **CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES** 64
- #define **CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES** 64
- #define **CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES** 3
- #define **CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES** 4
- #define **CC_UTIL_MAX_KDF_SIZE_IN_BYTES** (**CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES**+**CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES**+**CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES**)
- #define **CC_UTIL_MAX_DERIVED_KEY_SIZE_IN_BYTES** 4080

2.19.4.3 Macro Definition Documentation

2.19.4.3.1 #define CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES 4

The maximal size of the fixed data in bytes.

2.19.4.3.2 #define CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES 3

The minimal size of the fixed data in bytes.

2.19.4.3.3 #define CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES 64

The maximal length of the context in bytes.

2.19.4.3.4 #define CC_UTIL_MAX_DERIVED_KEY_SIZE_IN_BYTES 4080

The maximal size of the derived-key in bytes.

2.19.4.3.5 #define CC_UTIL_MAX_KDF_SIZE_IN_BYTES (CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES+CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES+CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES)

The maximal size of the derived-key material in bytes.

2.19.4.3.6 #define CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES 64

The maximal length of the label in bytes.

2.19.5 CryptoCell utility key-derivation APIs

Contains the CryptoCell utility key-derivation function API.

2.19.5.1 Modules

- **CryptoCell utility general key definitions**

  Contains KDF API definitions.

2.19.5.2 Files

- file **mbedtls_cc_util_key_derivation.h**

  This file contains the CryptoCell utility key-derivation function APIs.

2.19.5.3 Macros

- #define **mbedtls_util_key_derivation_cmac**(keyType, pUserKey, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) **mbedtls_util_key_derivation**(keyType, pUserKey, **CC_UTIL_PRF_CMAC**, **CC_HASH_OperationModeLast**, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

  This function performs key derivation using using AES-CMAC.

- #define **mbedtls_util_key_derivation_hmac**(keyType, pUserKey, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) **mbedtls_util_key_derivation**(keyType, pUserKey, **CC_UTIL_PRF_HMAC**, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

  This function performs key derivation using HMAC.

2.19.5.4 Enumerations

- enum **mbedtls_util_keytype_t** { **CC_UTIL_USER_KEY** = 0, **CC_UTIL_ROOT_KEY** = 1, **CC_UTIL_TOTAL_KEYS** = 2, **CC_UTIL_END_OF_KEY_TYPE** = 0x7FFFFFFF }
- enum **mbedtls_util_prftype_t** { **CC_UTIL_PRF_CMAC** = 0, **CC_UTIL_PRF_HMAC** = 1, **CC_UTIL_TOTAL_PRFS** = 2, **CC_UTIL_END_OF_PRF_TYPE** = 0x7FFFFFFF }

2.19.5.5 Functions

- **CCUtilError_t mbedtls_util_key_derivation** (**mbedtls_util_keytype_t** keyType, **mbedtls_util_keydata** *pUserKey, **mbedtls_util_prftype_t** prfType, **CCHashOperationMode_t** hashMode, const uint8_t *pLabel, size_t labelSize, const uint8_t *pContextData, size_t contextSize, uint8_t *pDerivedKey, size_t derivedKeySize)

This function performs key derivation.

2.19.5.6 Macro Definition Documentation

2.19.5.6.1 #define mbedtls_util_key_derivation_cmac(keyType, pUserKey, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) mbedtls_util_key_derivation(keyType, pUserKey, CC_UTIL_PRF_CMAC, CC_HASH_OperationModeLast, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

It is defined as specified in the KDF in Counter Mode section in *NIST SP 800-108: Recommendation for Key Derivation Using Pseudorandom Functions*.

The derivation is based on length l, label L, context C, and derivation key Ki.

Returns:

CC_UTIL_OK on success.

A non-zero value from **cc_util_error.h** on failure.

2.19.5.6.2 #define mbedtls_util_key_derivation_hmac(keyType, pUserKey, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) mbedtls_util_key_derivation(keyType, pUserKey, CC_UTIL_PRF_HMAC, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

It is defined as specified in the KDF in Counter Mode section in *NIST SP 800-108: Recommendation for Key Derivation Using Pseudorandom Functions*.

The derivation is based on length l, label L, context C, and derivation key Ki.

HMAC is used as the pseudo-random function (PRF).

Returns:

CC_UTIL_OK on success.

A non-zero value from **cc_util_error.h** on failure.

2.19.5.7 Enumeration Type Documentation

2.19.5.7.1 enum mbedtls_util_keytype_t

Derivation type of the input key.

Enumerator:

Enum	Description
`CC_UTIL_USER_KEY`	The user key.

Enum	Description
`CC_UTIL_ROOT_KEY`	The device root key (the HUK).
`CC_UTIL_TOTAL_KEYS`	Total number of keys.
`CC_UTIL_END_OF_KEY_TYPE`	Reserved.

2.19.5.7.2 enum mbedtls_util_prftype_t

Pseudo-random function type for key derivation.

Enumerator:

Enum	Description
`CC_UTIL_PRF_CMAC`	The CMAC function.
`CC_UTIL_PRF_HMAC`	The HMAC function.
`CC_UTIL_TOTAL_PRFS`	The total number of pseudo-random functions.
`CC_UTIL_END_OF_PRF_TYPE`	Reserved.

2.19.5.8 Function Documentation

2.19.5.8.1 CCUtilError_t **mbedtls_util_key_derivation (**mbedtls_util_keytype_t **keyType,** mbedtls_util_keydata * **pUserKey,** mbedtls_util_prftype_t **prfType,** CCHashOperationMode_t **hashMode, const uint8_t * pLabel, size_t labelSize, const uint8_t * pContextData, size_t contextSize, uint8_t * pDerivedKey, size_t derivedKeySize)**

It is defined as specified in the KDF in Counter Mode section in *NIST SP 800-108 Recommendation for Key Derivation Using Pseudorandom Functions*.

The derivation is based on length l, label L, context C, and derivation key Ki.

AES-CMAC or HMAC are used as the pseudo-random function (PRF).

You must define the label and context for each use-case well when using this API.

Returns:

CC_UTIL_OK on success.

A non-zero value from **cc_util_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`keyType`	The key type that is used as an input to a key-derivation function: CC_UTIL_USER_KEY or CC_UTIL_ROOT_KEY.

I/O	Parameter	Description
in	`pUserKey`	A pointer to the key buffer of the user, in case of CC_UTIL_USER_KEY.
in	`prfType`	The PRF type that is used as an input to a key-derivation function: CC_UTIL_PRF_CMAC or CC_UTIL_PRF_HMAC.
in	`hashMode`	One of the supported hash modes that are defined in CCHashOperationMode_t.
in	`pLabel`	A string that identifies the purpose for the derived keying material.
in	`labelSize`	The label size must be in range of 1 to 64 bytes in length.
in	`pContextData`	A binary string containing the information related to the derived keying material.
in	`contextSize`	The context size must be in range of 1 to 64 bytes in length.
out	`pDerivedKey`	Keying material output. Must be at least the size of derivedKeySize.
in	`derivedKeySize`	The size of the derived keying material in bytes, up to 4080 bytes.

2.19.6 Specific errors of the CryptoCell utility module APIs

Contains utility API error definitions.

2.19.6.1 Files

- file **cc_util_error.h**

  This file contains the error definitions of the CryptoCell utility APIs.

2.19.6.2 Macros

- #define **CC_UTIL_OK** 0x00UL
- #define **CC_UTIL_MODULE_ERROR_BASE** 0x80000000
- #define **CC_UTIL_INVALID_KEY_TYPE** (**CC_UTIL_MODULE_ERROR_BASE** + 0x00UL)
- #define **CC_UTIL_DATA_IN_POINTER_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x01UL)
- #define **CC_UTIL_DATA_IN_SIZE_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x02UL)
- #define **CC_UTIL_DATA_OUT_POINTER_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x03UL)
- #define **CC_UTIL_DATA_OUT_SIZE_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x04UL)
- #define **CC_UTIL_FATAL_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x05UL)
- #define **CC_UTIL_ILLEGAL_PARAMS_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x06UL)

- #define **CC_UTIL_BAD_ADDR_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x07UL)
- #define **CC_UTIL_EK_DOMAIN_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x08UL)
- #define **CC_UTIL_KDR_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x09UL)
- #define **CC_UTIL_LCS_INVALID_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x0AUL)
- #define **CC_UTIL_SESSION_KEY_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x0BUL)
- #define **CC_UTIL_INVALID_USER_KEY_SIZE** (**CC_UTIL_MODULE_ERROR_BASE** + 0x0DUL)
- #define **CC_UTIL_ILLEGAL_LCS_FOR_OPERATION_ERR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x0EUL)
- #define **CC_UTIL_INVALID_PRF_TYPE** (**CC_UTIL_MODULE_ERROR_BASE** + 0x0FUL)
- #define **CC_UTIL_INVALID_HASH_MODE** (**CC_UTIL_MODULE_ERROR_BASE** + 0x10UL)
- #define **CC_UTIL_UNSUPPORTED_HASH_MODE** (**CC_UTIL_MODULE_ERROR_BASE** + 0x11UL)
- #define **CC_UTIL_KEY_UNUSABLE_ERROR** (**CC_UTIL_MODULE_ERROR_BASE** + 0x12UL)

2.19.6.3 Macro Definition Documentation

2.19.6.3.1 #define CC_UTIL_BAD_ADDR_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x07UL)

Invalid address given.

2.19.6.3.2 #define CC_UTIL_DATA_IN_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x01UL)

Illegal data-in pointer.

2.19.6.3.3 #define CC_UTIL_DATA_IN_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x02UL)

Illegal data-in size.

2.19.6.3.4 #define CC_UTIL_DATA_OUT_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x03UL)

Illegal data-out pointer.

2.19.6.3.5 #define CC_UTIL_DATA_OUT_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x04UL)

Illegal data-out size.

2.19.6.3.6 #define CC_UTIL_EK_DOMAIN_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x08UL)

Illegal domain for endorsement key.

2.19.6.3.7 #define CC_UTIL_FATAL_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x05UL)

Fatal error.

2.19.6.3.8 #define CC_UTIL_ILLEGAL_LCS_FOR_OPERATION_ERR (CC_UTIL_MODULE_ERROR_BASE + 0x0EUL)

Illegal LCS for the required operation.

2.19.6.3.9 #define CC_UTIL_ILLEGAL_PARAMS_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x06UL)

Illegal parameters.

2.19.6.3.10 #define CC_UTIL_INVALID_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x10UL)

Invalid hash mode.

2.19.6.3.11 #define CC_UTIL_INVALID_KEY_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x00UL)

Illegal key type.

2.19.6.3.12 #define CC_UTIL_INVALID_PRF_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x0FUL)

Invalid PRF type.

2.19.6.3.13 #define CC_UTIL_INVALID_USER_KEY_SIZE (CC_UTIL_MODULE_ERROR_BASE + 0x0DUL)

Illegal user key size.

2.19.6.3.14 #define CC_UTIL_KDR_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x09UL)

HUK is not valid.

2.19.6.3.15 #define CC_UTIL_KEY_UNUSABLE_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x12UL)

Key is unusable

2.19.6.3.16 #define CC_UTIL_LCS_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0AUL)

LCS is not valid.

2.19.6.3.17 #define CC_UTIL_MODULE_ERROR_BASE 0x80000000

The error base address definition.

2.19.6.3.18 #define CC_UTIL_OK 0x00UL

Success definition.

2.19.6.3.19 #define CC_UTIL_SESSION_KEY_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0BUL)

Session key is not valid.

2.19.6.3.20 #define CC_UTIL_UNSUPPORTED_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x11UL)

Unsupported hash mode.

2.20 CryptoCell production-library APIs

Contains CryptoCell production-library APIs.

2.20.1 Modules

- **CryptoCell production-library definitions**

  Contains CryptoCell production-library definitions.

- **CryptoCell ICV production library APIs**

  Contains CryptoCell ICV production library APIs.
- **CryptoCell OEM production library APIs**

  Contains CryptoCell OEM production library APIs.
- **Specific errors of the CryptoCell production-library APIs**

  Contains the CryptoCell production-library-API error definitions.

2.20.2 CryptoCell production-library definitions

Contains CryptoCell production-library definitions.

2.20.2.1 Files

- file **cc_prod.h**

  This file contains all of the enums and definitions that are used for the ICV and OEM production libraries.

2.20.2.2 Data Structures

- union **CCAssetBuff_t**

  The asset buffer.

2.20.2.3 Macros

- #define **CC_PROD_32BIT_WORD_SIZE** sizeof(uint32_t)
- #define **PROD_ASSET_SIZE** 16
- #define **PROD_ASSET_PKG_SIZE** 64
- #define **PROD_ASSET_PKG_WORD_SIZE** (**PROD_ASSET_PKG_SIZE**/**CC_PROD_32BIT_WORD_SIZE**)
- #define **PROD_DCU_LOCK_WORD_SIZE** 4

2.20.2.4 Typedefs

- typedef uint8_t **CCPlainAsset_t**[**PROD_ASSET_SIZE**]
- typedef uint32_t **CCAssetPkg_t**[**PROD_ASSET_PKG_WORD_SIZE**]

2.20.2.5 Enumerations

- enum **CCAssetType_t** { **ASSET_NO_KEY** = 0, **ASSET_PLAIN_KEY** = 1, **ASSET_PKG_KEY** = 2, **ASSET_TYPE_RESERVED** = 0x7FFFFFFF }

2.20.2.6 Macro Definition Documentation

2.20.2.6.1 #define CC_PROD_32BIT_WORD_SIZE sizeof(uint32_t)

The definition of the number of bytes in a word.

2.20.2.6.2 #define PROD_ASSET_PKG_SIZE 64

The size of the asset-package in bytes.

2.20.2.6.3 #define PROD_ASSET_PKG_WORD_SIZE (PROD_ASSET_PKG_SIZE/CC_PROD_32BIT_WORD_SIZE)

The size of the asset-package in words.

2.20.2.6.4 #define PROD_ASSET_SIZE 16

The size of the plain-asset in bytes.

2.20.2.6.5 #define PROD_DCU_LOCK_WORD_SIZE 4

The number of words of the DCU lock.

2.20.2.7 Typedef Documentation

2.20.2.7.1 typedef uint32_t CCAssetPkg_t[PROD_ASSET_PKG_WORD_SIZE]

Defines the buffer of the asset-package, as a 64-byte array.

2.20.2.7.2 typedef uint8_t CCPlainAsset_t[PROD_ASSET_SIZE]

Defines the buffer of the plain asset, as a 16-byte array.

2.20.2.8 Enumeration Type Documentation

2.20.2.8.1 enum CCAssetType_t

The type of the provided asset.

Enumerator:

Enum	Description
`ASSET_NO_KEY`	The asset is not provided.
`ASSET_PLAIN_KEY`	The asset is provided as plain, not in a package.
`ASSET_PKG_KEY`	The asset is provided as a package.

Enum	Description
ASSET_TYPE_RESERVED	Reserved.

2.20.3 CryptoCell ICV production library APIs

Contains CryptoCell ICV production library APIs.

2.20.3.1 Files

- file **cc_cmpu.h**

  This file contains all of the ICV production library APIs, their enums and definitions.

2.20.3.2 Data Structures

- union **CCCmpuUniqueBuff_t**

  The device use of the unique buffer.

- struct **CCCmpuData_t**

2.20.3.3 Macros

- #define **CMPU_WORKSPACE_MINIMUM_SIZE** 4096
- #define **PROD_UNIQUE_BUFF_SIZE** 16

2.20.3.4 Enumerations

- enum **CCCmpuUniqueDataType_t** { **CMPU_UNIQUE_IS_HBK0** = 1, **CMPU_UNIQUE_IS_USER_DATA** = 2, **CMPU_UNIQUE_RESERVED** = 0x7FFFFFFF }

2.20.3.5 Functions

- CIMPORT_C CCError_t **CCProd_Cmpu** (unsigned long ccHwRegBaseAddr, **CCCmpuData_t** *pCmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all ICV assets into the OTP of the device.

2.20.3.6 Macro Definition Documentation

2.20.3.6.1 #define CMPU_WORKSPACE_MINIMUM_SIZE 4096

The size of the ICV production library workspace in bytes. This workspace is needed by the library for internal use.

2.20.3.6.2 #define PROD_UNIQUE_BUFF_SIZE 16

The size of the ICV production library unique buffer in bytes: Hbk0 or user data.

2.20.3.7 Enumeration Type Documentation

2.20.3.7.1 enum CCCmpuUniqueDataType_t

The unique data type.

Enumerator:

Enum	Description
`CMPU_UNIQUE_IS_HBK0`	The device uses the unique data as Hbk0.
`CMPU_UNIQUE_IS_USER_DATA`	The device uses the unique data as a random value. Hbk0 is not used for the device.
`CMPU_UNIQUE_RESERVED`	Reserved.

2.20.3.8 Function Documentation

2.20.3.8.1 CIMPORT_C CCError_t CCProd_Cmpu (unsigned long ccHwRegBaseAddr, CCCmpuData_t * pCmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

The user must perform a power-on-reset (PoR) to trigger LCS change to DM LCS.

Returns:

CC_OK on success.

A non-zero value from **cc_prod_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`ccHwRegBaseAddr`	The base address of CrytoCell HW registers.
in	`pCmpuData`	A pointer to the ICV defines structure.
in	`workspaceBaseAddr`	The base address of the workspace for internal use.
in	`workspaceSize`	The size of the provided workspace. Must be at least **CMPU_WORKSPACE_MINIMUM_SIZE**.

2.20.4 CryptoCell OEM production library APIs

Contains CryptoCell OEM production library APIs.

2.20.4.1 Files

- file **cc_dmpu.h**

  This file contains all of the OEM production library APIs, their enums and definitions.

2.20.4.2 Data Structures

- union **CCDmpuHbkBuff_t**
- struct **CCDmpuData_t**

2.20.4.3 Macros

- #define **DMPU_WORKSPACE_MINIMUM_SIZE** 1536
- #define **DMPU_HBK1_SIZE_IN_WORDS** 4
- #define **DMPU_HBK_SIZE_IN_WORDS** 8

2.20.4.4 Enumerations

- enum **CCDmpuHBKType_t** { **DMPU_HBK_TYPE_HBK1** = 1, **DMPU_HBK_TYPE_HBK** = 2, **DMPU_HBK_TYPE_RESERVED** = 0x7FFFFFFF }

2.20.4.5 Functions

- CIMPORT_C CCError_t **CCProd_Dmpu** (unsigned long ccHwRegBaseAddr, **CCDmpuData_t** *pDmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all OEM assets into the OTP of the device.

2.20.4.6 Macro Definition Documentation

2.20.4.6.1 #define DMPU_HBK1_SIZE_IN_WORDS 4

The size of the Hbk1 buffer in words.

2.20.4.6.2 #define DMPU_HBK_SIZE_IN_WORDS 8

The size of the Hbk buffer in words.

2.20.4.6.3 #define DMPU_WORKSPACE_MINIMUM_SIZE 1536

The size of the OEM production library workspace in bytes. This workspace is required by the library for internal use.

2.20.4.7 Enumeration Type Documentation

2.20.4.7.1 enum CCDmpuHBKType_t

The type of the unique data.

Enumerator:

Enum	Description
`DMPU_HBK_TYPE_HBK1`	The device uses Hbk1.
`DMPU_HBK_TYPE_HBK`	The device uses a full Hbk.
`DMPU_HBK_TYPE_RESERVED`	Reserved.

2.20.4.8 Function Documentation

2.20.4.8.1 CIMPORT_C CCError_t CCProd_Dmpu (unsigned long ccHwRegBaseAddr, CCDmpuData_t * pDmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

The user must perform a power-on-reset (PoR) to trigger LCS change to Secure.

Returns:

CC_OK on success.

A non-zero value from **cc_prod_error.h** on failure.

Parameters:

I/O	Parameter	Description
in	`ccHwRegBaseAddr`	The base address of CrytoCell HW registers.
in	`pDmpuData`	A pointer to the defines structure of the OEM.
in	`workspaceBaseAddr`	The base address of the workspace for internal use.
in	`workspaceSize`	The size of provided workspace. Must be at least DMPU_WORKSPACE_MINIMUM_SIZE.

2.20.5 Specific errors of the CryptoCell production-library APIs

Contains the CryptoCell production-library-API error definitions.

2.20.5.1 Files

- file **cc_prod_error.h**

  This file contains the error definitions of the CryptoCell production-library APIs.

2.20.5.2 Macros

- #define **CC_PROD_INIT_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x01UL)
- #define **CC_PROD_INVALID_PARAM_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x02UL)
- #define **CC_PROD_ILLEGAL_ZERO_COUNT_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x03UL)
- #define **CC_PROD_ILLEGAL_LCS_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x04UL)
- #define **CC_PROD_ASSET_PKG_PARAM_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x05UL)
- #define **CC_PROD_ASSET_PKG_VERIFY_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x06UL)
- #define **CC_PROD_HAL_FATAL_ERR** (**CC_PROD_MODULE_ERROR_BASE** + 0x07UL)

2.20.5.3 Macro Definition Documentation

2.20.5.3.1 #define CC_PROD_ASSET_PKG_PARAM_ERR (CC_PROD_MODULE_ERROR_BASE + 0x05UL)

Invalid asset-package fields.

2.20.5.3.2 #define CC_PROD_ASSET_PKG_VERIFY_ERR (CC_PROD_MODULE_ERROR_BASE + 0x06UL)

Failed to validate the asset package.

2.20.5.3.3 #define CC_PROD_HAL_FATAL_ERR (CC_PROD_MODULE_ERROR_BASE + 0x07UL)

HAL Fatal error occured.

2.20.5.3.4 #define CC_PROD_ILLEGAL_LCS_ERR (CC_PROD_MODULE_ERROR_BASE + 0x04UL)

LCS is invalid for the operation.

2.20.5.3.5 #define CC_PROD_ILLEGAL_ZERO_COUNT_ERR (CC_PROD_MODULE_ERROR_BASE + 0x03UL)

Invalid number of zeroes calculated.

2.20.5.3.6 #define CC_PROD_INIT_ERR (CC_PROD_MODULE_ERROR_BASE + 0x01UL)

Library initialization failure.

2.20.5.3.7 #define CC_PROD_INVALID_PARAM_ERR (CC_PROD_MODULE_ERROR_BASE + 0x02UL)

Illegal parameter.

2.21 Data structure documentation

2.21.1 CC_PalTrngParams_t struct reference

```
#include <cc_pal_trng.h>
```

2.21.1.1 Data Fields

- uint32_t **SubSamplingRatio1**
- uint32_t **SubSamplingRatio2**
- uint32_t **SubSamplingRatio3**
- uint32_t **SubSamplingRatio4**

2.21.1.2 Detailed Description

Definition for the structure of the random-generator parameters of CryptoCell, containing the user-given parameters.

2.21.1.3 Field Documentation

2.21.1.3.1 uint32_t CC_PalTrngParams_t::SubSamplingRatio1

The sampling ratio of ROSC #1.

2.21.1.3.2 uint32_t CC_PalTrngParams_t::SubSamplingRatio2

The sampling ratio of ROSC #2.

2.21.1.3.3 uint32_t CC_PalTrngParams_t::SubSamplingRatio3

The sampling ratio of ROSC #3.

2.21.1.3.4 uint32_t CC_PalTrngParams_t::SubSamplingRatio4

The sampling ratio of ROSC #4.

The documentation for this struct was generated from the following file:

- cc_pal_trng.h

2.21.2 CCAesHwKeyData_t struct reference

```
#include <cc_aes_defs.h>
```

2.21.2.1 Data Fields

- size_t **slotNumber**

2.21.2.2 Detailed Description

The AES HW key Data.

2.21.2.3 Field Documentation

2.21.2.3.1 size_t CCAesHwKeyData_t::slotNumber

Slot number.

The documentation for this struct was generated from the following file:

- cc_aes_defs.h

2.21.3 CCAesUserContext_t struct reference

```
#include <cc_aes_defs.h>
```

2.21.3.1 Data Fields

- uint32_t **buff** [**CC_AES_USER_CTX_SIZE_IN_WORDS**]

2.21.3.2 Detailed Description

The context prototype of the user.

The argument type that is passed by the user to the AES APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.21.3.3 Field Documentation

2.21.3.3.1 uint32_t CCAesUserContext_t::buff[CC_AES_USER_CTX_SIZE_IN_WORDS]

The context buffer for internal usage.

The documentation for this struct was generated from the following file:

- cc_aes_defs.h

2.21.4 CCAesUserKeyData_t struct reference

```
#include <cc_aes_defs.h>
```

2.21.4.1 Data Fields

- uint8_t ***pKey**
- size_t **keySize**

2.21.4.2 Detailed Description

The AES key data of the user.

2.21.4.3 Field Documentation

2.21.4.3.1 size_t CCAesUserKeyData_t::keySize

The size of the key in bytes. Valid values for XTS mode, if supported: 32 bytes or 64 bytes, indicating the full size of the double key (2x128 or 2x256 bit). Valid values for XCBC-MAC mode: 16 bytes, as limited by the standard. Valid values for all other modes: 16 bytes, 24 bytes, or 32 bytes.

2.21.4.3.2 uint8_t*CCAesUserKeyData_t::pKey

A pointer to the key.

The documentation for this struct was generated from the following file:

- cc_aes_defs.h

2.21.5 CCAssetBuff_t union reference

The asset buffer.

```
#include <cc_prod.h>
```

2.21.5.1 Data Fields

- **CCPlainAsset_t plainAsset**
- **CCAssetPkg_t pkgAsset**

2.21.5.2 Detailed Description

If the asset is provided as plain asset, the plainAsset field is used. Otherwise, the pkgAsset field is used.

2.21.5.3 Field Documentation

2.21.5.3.1 CCAssetPkg_t **CCAssetBuff_t::pkgAsset**

Asset-package buffer.

2.21.5.3.2 CCPlainAsset_t **CCAssetBuff_t::plainAsset**

Plain asset buffer.

The documentation for this union was generated from the following file:

- **cc_prod.h**

2.21.6 CCCmpuData_t struct reference

```
#include <cc_cmpu.h>
```

2.21.6.1 Data Fields

- **CCCmpuUniqueDataType_t uniqueDataType**
- **CCCmpuUniqueBuff_t uniqueBuff**
- **CCAssetType_t kpicvDataType**
- **CCAssetBuff_t kpicv**
- **CCAssetType_t kceicvDataType**
- **CCAssetBuff_t kceicv**
- uint32_t **icvMinVersion**
- uint32_t **icvConfigWord**
- uint32_t **icvDcuDefaultLock** [**PROD_DCU_LOCK_WORD_SIZE**]

2.21.6.2 Detailed Description

The ICV production library input options.

2.21.6.3 Field Documentation

2.21.6.3.1 uint32_t CCCmpuData_t::icvConfigWord

The ICV configuration word.

2.21.6.3.2 uint32_t CCCmpuData_t::icvDcuDefaultLock[PROD_DCU_LOCK_WORD_SIZE]

The default DCU lock bits of the ICV. Valid only if Hbk0 is used.

2.21.6.3.3 uint32_t CCCmpuData_t::icvMinVersion

The minimal SW version of the ICV. Valid only if Hbk0 is used.

2.21.6.3.4 CCAssetBuff_t **CCCmpuData_t::kceicv**

The Kceicv buffer, if its type is Plain-asset or Package.

2.21.6.3.5 CCAssetType_t **CCCmpuData_t::kceicvDataType**

The asset type of the Kceicv. Allowed values are: Not used, Plain-asset, or Package.

2.21.6.3.6 CCAssetBuff_t **CCCmpuData_t::kpicv**

The Kpicv buffer, if its type is Plain-asset or Package.

2.21.6.3.7 CCAssetType_t **CCCmpuData_t::kpicvDataType**

The Kpicv asset type. Allowed values are: Not used, Plain-asset, or Package.

2.21.6.3.8 CCCmpuUniqueBuff_t **CCCmpuData_t::uniqueBuff**

The unique data buffer.

2.21.6.3.9 CCCmpuUniqueDataType_t **CCCmpuData_t::uniqueDataType**

The unique data type: Hbk0 or a random user-defined data.

The documentation for this struct was generated from the following file:

- **cc_cmpu.h**

2.21.7 CCCmpuUniqueBuff_t union reference

The device use of the unique buffer.

```
#include <cc_cmpu.h>
```

2.21.7.1 Data Fields

- uint8_t **hbk0** [**PROD_UNIQUE_BUFF_SIZE**]
- uint8_t **userData** [**PROD_UNIQUE_BUFF_SIZE**]

2.21.7.2 Detailed Description

If the device uses Hbk0, then the hbk0 field is used. Otherwise, a random buffer for the userData field is used.

2.21.7.3 Field Documentation

2.21.7.3.1 uint8_t CCCmpuUniqueBuff_t::hbk0[PROD_UNIQUE_BUFF_SIZE]

The Hbk0 buffer, if used by the device.

2.21.7.3.2 uint8_t CCCmpuUniqueBuff_t::userData[PROD_UNIQUE_BUFF_SIZE]

Any random value, if Hbk0 is not used by the device.

The documentation for this union was generated from the following file:

- cc_cmpu.h

2.21.8 CCDmpuData_t struct reference

```
#include <cc_dmpu.h>
```

2.21.8.1 Data Fields

- **CCDmpuHBKType_t hbkType**
- **CCDmpuHbkBuff_t hbkBuff**
- **CCAssetType_t kcpDataType**
- **CCAssetBuff_t kcp**
- **CCAssetType_t kceDataType**
- **CCAssetBuff_t kce**
- uint32_t **oemMinVersion**
- uint32_t **oemDcuDefaultLock** [**PROD_DCU_LOCK_WORD_SIZE**]

2.21.8.2 Detailed Description

The OEM production library input defines.

2.21.8.3 Field Documentation

2.21.8.3.1 CCDmpuHbkBuff_t CCDmpuData_t::hbkBuff

The Hbk buffer.

2.21.8.3.2 CCDmpuHBKType_t CCDmpuData_t::hbkType

The type of Hbk: Hbk1 - 128 bits. Hbk - 256 bits.

2.21.8.3.3 CCAssetBuff_t **CCDmpuData_t::kce**

The Kce buffer, if kceDataType is Plain-asset or package.

2.21.8.3.4 CCAssetType_t **CCDmpuData_t::kceDataType**

The Kce asset type: Not used, Plain-asset, or Package.

2.21.8.3.5 CCAssetBuff_t **CCDmpuData_t::kcp**

The Kcp buffer, if kcpDataType is Plain-asset or package.

2.21.8.3.6 CCAssetType_t **CCDmpuData_t::kcpDataType**

The Kcp asset type: Not used, Plain-asset, or Package.

2.21.8.3.7 uint32_t **CCDmpuData_t::oemDcuDefaultLock**[PROD_DCU_LOCK_WORD_SIZE]

The default DCU lock bits of the OEM.

2.21.8.3.8 uint32_t **CCDmpuData_t::oemMinVersion**

The minimal SW version of the OEM.

The documentation for this struct was generated from the following file:

- o cc_dmpu.h

2.21.9 CCDmpuHbkBuff_t union reference

```
#include <cc_dmpu.h>
```

2.21.9.1 Data Fields

- uint8_t **hbk1** [**DMPU_HBK1_SIZE_IN_WORDS** *__CC_PROD_32BIT_WORD_SIZE__]
- uint8_t **hbk** [**DMPU_HBK_SIZE_IN_WORDS** *__CC_PROD_32BIT_WORD_SIZE__]

2.21.9.2 Detailed Description

The device use of the Hbk buffer.

If the device uses Hbk0 and Hbk1, then the Hbk1 field is used. Otherwise, the Hbk field is used.

2.21.9.3 Field Documentation

2.21.9.3.1 uint8_t CCDmpuHbkBuff_t::hbk[DMPU_HBK_SIZE_IN_WORDS *CC_PROD_32BIT_WORD_SIZE]

The full 256-bit Hbk buffer.

2.21.9.3.2 uint8_t CCDmpuHbkBuff_t::hbk1[DMPU_HBK1_SIZE_IN_WORDS *CC_PROD_32BIT_WORD_SIZE]

The Hbk1 buffer, if used by the device.

The documentation for this union was generated from the following file:

- **cc_dmpu.h**

2.21.10 CCEcdhFipsKatContext_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.10.1 Data Fields

- **CCEcpkiUserPublKey_t pubKey**
- **CCEcpkiUserPrivKey_t privKey**
- union {

  CCEcpkiBuildTempData_t ecpkiTempData

  CCEcdhTempData_t ecdhTempBuff

  } **tmpData**
- uint8_t **secretBuff** [**CC_ECPKI_FIPS_ORDER_LENGTH**]

2.21.10.2 Detailed Description

ECDH KAT data structures for FIPS certification.

2.21.10.3 Field Documentation

2.21.10.3.1 CCEcpkiUserPrivKey_t CCEcdhFipsKatContext_t::privKey

The private key data.

2.21.10.3.2 CCEcpkiUserPublKey_t CCEcdhFipsKatContext_t::pubKey

The public key data.

2.21.10.3.3 uint8_t CCEcdhFipsKatContext_t::secretBuff[CC_ECPKI_FIPS_ORDER_LENGTH]

The buffer for the secret key.

2.21.10.3.4 union { ... } CCEcdhFipsKatContext_t::tmpData

Internal buffers.

The documentation for this struct was generated from the following file:

- o cc_ecpki_types.h

2.21.11 CCEcdhTempData_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.11.1 Data Fields

- uint32_t **ccEcdhIntBuff** [**CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS**]

2.21.11.2 Detailed Description

The type of the ECDH temporary data.

2.21.11.3 Field Documentation

2.21.11.3.1 uint32_t CCEcdhTempData_t::ccEcdhIntBuff[CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS]

Temporary buffers.

The documentation for this struct was generated from the following file:

- o cc_ecpki_types.h

2.21.12 CCEcdsaFipsKatContext_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.12.1 Data Fields

- union {

    struct {

    CCEcpkiUserPrivKey_t PrivKey

    CCEcdsaSignUserContext_t signCtx

    } **userSignData**

    struct {

    CCEcpkiUserPublKey_t PublKey

        union {

        CCEcdsaVerifyUserContext_t verifyCtx

        CCEcpkiBuildTempData_t tempData

        } buildOrVerify

    } **userVerifyData**

- } **keyContextData**
- uint8_t **signBuff** [2 ***CC_ECPKI_FIPS_ORDER_LENGTH**]

2.21.12.2 Detailed Description

ECDSA KAT data structures for FIPS certification. The ECDSA KAT tests are defined for domain 256r1.

2.21.12.3 Field Documentation

2.21.12.3.1 union { ... } CCEcdsaFipsKatContext_t::keyContextData

The key data.

2.21.12.3.2 uint8_t CCEcdsaFipsKatContext_t::signBuff[2 *CC_ECPKI_FIPS_ORDER_LENGTH]

Internal buffer.

2.21.12.3.3 struct { ... } CCEcdsaFipsKatContext_t::userSignData

The private key data.

2.21.12.3.4 struct { ... } CCEcdsaFipsKatContext_t::userVerifyData

The public key data.

The documentation for this struct was generated from the following file:

- cc_ecpki_types.h

2.21.13 CCEcdsaSignUserContext_t struct reference

The context definition of the user for the signing operation.

```
#include <cc_ecpki_types.h>
```

2.21.13.1 Data Fields

- uint32_t **context_buff** [(sizeof(**EcdsaSignContext_t**)+3)/4]
- uint32_t **valid_tag**

2.21.13.2 Detailed Description

This context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.21.13.3 Field Documentation

2.21.13.3.1 uint32_t CCEcdsaSignUserContext_t::context_buff[(sizeof(EcdsaSignContext_t)+3)/4]

The data of the signing process.

2.21.13.3.2 uint32_t CCEcdsaSignUserContext_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- cc_ecpki_types.h

2.21.14 CCEcdsaVerifyUserContext_t struct reference

The context definition of the user for the verification operation.

```
#include <cc_ecpki_types.h>
```

2.21.14.1 Data Fields

- uint32_t **context_buff** [(sizeof(**EcdsaVerifyContext_t**)+3)/4]
- uint32_t **valid_tag**

2.21.14.2 Detailed Description

The context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.21.14.3 Field Documentation

2.21.14.3.1 uint32_t CCEcdsaVerifyUserContext_t::context_buff[(sizeof(EcdsaVerifyContext_t)+3)/4]

The data of the verification process.

2.21.14.3.2 uint32_t CCEcdsaVerifyUserContext_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- o **cc_ecpki_types.h**

2.21.15 CCEciesTempData_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.15.1 Data Fields

- **CCEcpkiUserPrivKey_t PrivKey**
- **CCEcpkiUserPublKey_t PublKey**
- **CCEcpkiUserPublKey_t ConvPublKey**
- uint32_t **zz** [3 ***CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**+1]
- union {

  CCEcpkiBuildTempData_t buildTempbuff

  CCEcpkiKgTempData_t KgTempBuff

  CCEcdhTempData_t DhTempBuff

  } **tmp**

2.21.15.2 Detailed Description

The temporary data definition of the ECIES.

2.21.15.3 Field Documentation

2.21.15.3.1 CCEcpkiUserPublKey_t **CCEciesTempData_t::ConvPublKey**

The public-key data used by conversion from Mbed TLS to CryptoCell.

2.21.15.3.2 CCEcpkiUserPrivKey_t **CCEciesTempData_t::PrivKey**

The data of the private key.

2.21.15.3.3 CCEcpkiUserPublKey_t **CCEciesTempData_t::PublKey**

The data of the public key.

2.21.15.3.4 union { ... } CCEciesTempData_t::tmp

Internal buffers.

2.21.15.3.5 uint32_t CCEciesTempData_t::zz[3 *CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]

Internal buffer.

The documentation for this struct was generated from the following file:

- o **cc_ecpki_types.h**

2.21.16 CCEcpkiBuildTempData_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.16.1 Data Fields

- uint32_t **ccBuildTmpIntBuff** [**CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS**]

2.21.16.2 Detailed Description

EC build temporary data.

2.21.16.3 Field Documentation

2.21.16.3.1 uint32_t CCEcpkiBuildTempData_t::ccBuildTmpIntBuff[CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS]

Temporary buffers.

The documentation for this struct was generated from the following file:

- cc_ecpki_types.h

2.21.17 CCEcpkiDomain_t struct reference

The structure containing the EC domain parameters in little-endian form.

```
#include <cc_ecpki_types.h>
```

2.21.17.1 Data Fields

- uint32_t **ecP** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]
- uint32_t **ecA** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]
- uint32_t **ecB** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]
- uint32_t **ecR** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]
- uint32_t **ecGx** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]
- uint32_t **ecGy** [CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]
- uint32_t **ecH**
- uint32_t **llfBuff** [CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS]
- uint32_t **modSizeInBits**
- uint32_t **ordSizeInBits**
- uint32_t **barrTagSizeInWords**
- CCEcpkiDomainID_t **DomainID**
- int8_t **name** [20]

2.21.17.2 Detailed Description

EC equation: Y^2 = X^3 + A*X + B over prime field GFp.

2.21.17.3 Field Documentation

2.21.17.3.1 uint32_t CCEcpkiDomain_t::barrTagSizeInWords

The size of each inserted Barret tag in words. Zero if not inserted.

2.21.17.3.2 CCEcpkiDomainID_t CCEcpkiDomain_t::DomainID

The EC Domain identifier.

2.21.17.3.3 uint32_t CCEcpkiDomain_t::ecA[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC equation parameter A.

2.21.17.3.4 uint32_t CCEcpkiDomain_t::ecB[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC equation parameter B.

2.21.17.3.5 uint32_t CCEcpkiDomain_t::ecGx[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

2.21.17.3.6 uint32_t CCEcpkiDomain_t::ecGy[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

2.21.17.3.7 uint32_t CCEcpkiDomain_t::ecH

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

2.21.17.3.8 uint32_t CCEcpkiDomain_t::ecP[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC modulus: P.

2.21.17.3.9 uint32_t CCEcpkiDomain_t::ecR[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]

Order of generator.

2.21.17.3.10 uint32_t CCEcpkiDomain_t::llfBuff[CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS]

Specific fields that are used by the low-level functions.

2.21.17.3.11 uint32_t CCEcpkiDomain_t::modSizeInBits

The size of fields in bits.

2.21.17.3.12 int8_t CCEcpkiDomain_t::name[20]

Internal buffer.

2.21.17.3.13 uint32_t CCEcpkiDomain_t::ordSizeInBits

The size of the order in bits.

The documentation for this struct was generated from the following file:

- o **cc_ecpki_types.h**

2.21.18 CCEcpkiKgFipsContext_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.18.1 Data Fields

- union {

  CCEcdsaSignUserContext_t signCtx

  CCEcdsaVerifyUserContext_t verifyCtx

  } **operationCtx**
- uint32_t **signBuff** [2 ***CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS**]

2.21.18.2 Detailed Description

ECPKI data structures for FIPS certification.

2.21.18.3 Field Documentation

2.21.18.3.1 union { ... } CCEcpkiKgFipsContext_t::operationCtx

Signing and verification data.

2.21.18.3.2 uint32_t CCEcpkiKgFipsContext_t::signBuff[2 *CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- o **cc_ecpki_types.h**

2.21.19 CCEcpkiKgTempData_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.19.1 Data Fields

- uint32_t **ccKGIntBuff** [**CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS**]

2.21.19.2 Detailed Description

The temporary data type of the ECPKI KG.

2.21.19.3 Field Documentation

2.21.19.3.1 uint32_t CCEcpkiKgTempData_t::ccKGIntBuff[CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- o cc_ecpki_types.h

2.21.20 CCEcpkiPointAffine_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.20.1 Data Fields

- uint32_t **x** [**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**]
- uint32_t **y** [**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**]

2.21.20.2 Detailed Description

The structure containing the EC point in affine coordinates and little endian form.

2.21.20.3 Field Documentation

2.21.20.3.1 uint32_t CCEcpkiPointAffine_t::x[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The X coordinate of the point.

2.21.20.3.2 uint32_t CCEcpkiPointAffine_t::y[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The Y coordinate of the point.

The documentation for this struct was generated from the following file:

- o cc_ecpki_types.h

2.21.21 CCEcpkiPrivKey_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.21.1 Data Fields

- uint32_t **PrivKey** [**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**+1]
- **CCEcpkiDomain_t domain**
- **CCEcpkiScaProtection_t scaProtection**

2.21.21.2 Detailed Description

The structure containing the data of the private key.

2.21.21.3 Field Documentation

2.21.21.3.1 CCEcpkiDomain_t **CCEcpkiPrivKey_t::domain**

The EC domain.

2.21.21.3.2 uint32_t CCEcpkiPrivKey_t::PrivKey[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]

The data of the private key.

2.21.21.3.3 CCEcpkiScaProtection_t **CCEcpkiPrivKey_t::scaProtection**

The SCA protection mode.

The documentation for this struct was generated from the following file:

- cc_ecpki_types.h

2.21.22 CCEcpkiPublKey_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.22.1 Data Fields

- uint32_t **x** [**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**]
- uint32_t **y** [**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS**]
- **CCEcpkiDomain_t domain**
- uint32_t **pointType**

2.21.22.2 Detailed Description

The structure containing the public key in affine coordinates.

2.21.22.3 Field Documentation

2.21.22.3.1 CCEcpkiDomain_t **CCEcpkiPublKey_t::domain**

The EC Domain.

2.21.22.3.2 uint32_t CCEcpkiPublKey_t::pointType

The point type.

2.21.22.3.3 uint32_t CCEcpkiPublKey_t::x[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The X coordinate of the public key.

2.21.22.3.4 uint32_t CCEcpkiPublKey_t::y[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The Y coordinate of the public key.

The documentation for this struct was generated from the following file:

- cc_ecpki_types.h

2.21.23 CCEcpkiUserPrivKey_t struct reference

The user structure prototype of the EC private key.

`#include <cc_ecpki_types.h>`

2.21.23.1 Data Fields

- uint32_t **valid_tag**
- uint32_t **PrivKeyDbBuff** [(sizeof(**CCEcpkiPrivKey_t**)+3)/4]

2.21.23.2 Detailed Description

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaSign().

2.21.23.3 Field Documentation

2.21.23.3.1 uint32_t CCEcpkiUserPrivKey_t::PrivKeyDbBuff[(sizeof(CCEcpkiPrivKey_t)+3)/4]

The data of the private key.

2.21.23.3.2 uint32_t CCEcpkiUserPrivKey_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- **cc_ecpki_types.h**

2.21.24 CCEcpkiUserPublKey_t struct reference

The user structure prototype of the EC public key.

```
#include <cc_ecpki_types.h>
```

2.21.24.1 Data Fields

- uint32_t **valid_tag**
- uint32_t **PublKeyDbBuff** [(sizeof(**CCEcpkiPublKey_t**)+3)/4]

2.21.24.2 Detailed Description

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaVerify().

2.21.24.3 Field Documentation

2.21.24.3.1 uint32_t CCEcpkiUserPublKey_t::PublKeyDbBuff[(sizeof(CCEcpkiPublKey_t)+3)/4]

The data of the public key.

2.21.24.3.2 uint32_t CCEcpkiUserPublKey_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- **cc_ecpki_types.h**

2.21.25 CCHashUserContext_t struct reference

```
#include <cc_hash_defs.h>
```

2.21.25.1 Data Fields

- uint32_t **buff** [**CC_HASH_USER_CTX_SIZE_IN_WORDS**]

2.21.25.2 Detailed Description

The context prototype of the user. The argument type that is passed by the user to the hash APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

2.21.25.3 Field Documentation

2.21.25.3.1 uint32_t CCHashUserContext_t::buff[CC_HASH_USER_CTX_SIZE_IN_WORDS]

The internal buffer.

The documentation for this struct was generated from the following file:

- o **cc_hash_defs.h**

2.21.26 CCRndContext_t struct reference

```
#include <cc_rnd_common.h>
```

2.21.26.1 Data Fields

- void ***rndState**
- void ***entropyCtx**
- **CCRndGenerateVectWorkFunc_t rndGenerateVectFunc**

2.21.26.2 Detailed Description

The definition of the RND context that includes the CryptoCell RND state structure, and a function pointer for the RND-generation function.

2.21.26.3 Field Documentation

2.21.26.3.1 void*CCRndContext_t::entropyCtx

A pointer to the entropy context. Note: This pointer should be allocated and assigned before calling **CC_LibInit()**.

2.21.26.3.2 CCRndGenerateVectWorkFunc_t CCRndContext_t::rndGenerateVectFunc

A pointer to the user-given function for generation a random vector.

2.21.26.3.3 void*CCRndContext_t::rndState

A pointer to the internal state of the RND. Note: This pointer should be allocated in a physical and contiguous memory, that is accessible to the CryptoCell DMA. This pointer should be allocated and assigned before calling **CC_LibInit()**.

The documentation for this struct was generated from the following file:

- cc_rnd_common.h

2.21.27 CCRndState_t struct reference

The structure for the RND state. This includes internal data that must be saved by the user between boots.

```
#include <cc_rnd_common.h>
```

2.21.27.1 Data Fields

- uint32_t **TrngProcesState**

2.21.27.2 Field Documentation

2.21.27.2.1 uint32_t CCRndState_t::TrngProcesState

The TRNG process state used internally in the code.

The documentation for this struct was generated from the following file:

- cc_rnd_common.h

2.21.28 CCRndWorkBuff_t struct reference

```
#include <cc_rnd_common.h>
```

2.21.28.1 Data Fields

- uint32_t **ccRndIntWorkBuff** [**CC_RND_WORK_BUFFER_SIZE_WORDS**]

2.21.28.2 Detailed Description

The definition of the RAM buffer, for internal use in instantiation or reseeding operations.

2.21.28.3 Field Documentation

2.21.28.3.1 uint32_t CCRndWorkBuff_t::ccRndIntWorkBuff[CC_RND_WORK_BUFFER_SIZE_WORDS]

The internal buffer.

The documentation for this struct was generated from the following file:

- o cc_rnd_common.h

2.21.29 CCSbCertInfo_t struct reference

```
#include <secureboot_defs.h>
```

2.21.29.1 Data Fields

- uint32_t **otpVersion**
- CCSbPubKeyIndexType_t **keyIndex**
- uint32_t **activeMinSwVersionVal**
- CCHashResult_t **pubKeyHash**
- uint32_t **initDataFlag**

2.21.29.2 Detailed Description

Input or output structure for the Secure Boot verification API.

2.21.29.3 Field Documentation

2.21.29.3.1 uint32_t CCSbCertInfo_t::activeMinSwVersionVal

The value of the SW version for the certificate-chain.

2.21.29.3.2 uint32_t CCSbCertInfo_t::initDataFlag

Internal flag for the initialization indication.

2.21.29.3.3 CCSbPubKeyIndexType_t CCSbCertInfo_t::keyIndex

The key hash to retrieve: The 128-bit Hbk0, the 128-bit Hbk1, or the 256-bit Hbk.

2.21.29.3.4 uint32_t CCSbCertInfo_t::otpVersion

The NV counter saved in OTP.

2.21.29.3.5 CCHashResult_t CCSbCertInfo_t::pubKeyHash

In: The hash of the public key (N||Np), to compare to the public key stored in the certificate.
Out: The hash of the public key (N||Np) stored in the certificate, to be used for verification of the public key of the next certificate in the chain.

The documentation for this struct was generated from the following file:

- o secureboot_defs.h

2.21.30 EcdsaSignContext_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.30.1 Data Fields

- **CCEcpkiUserPrivKey_t ECDSA_SignerPrivKey**
- **mbedtls_md_context_t hash_ctx**
- **CCHashResultBuf_t hashResult**
- uint32_t **hashResultSizeWords**
- **CCEcpkiHashOpMode_t hashMode**
- **CCEcdsaSignIntBuff_t ecdsaSignIntBuff**

2.21.30.2 Detailed Description

The context definition for the signing operation.

2.21.30.3 Field Documentation

2.21.30.3.1 CCEcpkiUserPrivKey_t **EcdsaSignContext_t::ECDSA_SignerPrivKey**

The data of the private key.

2.21.30.3.2 CCEcdsaSignIntBuff_t **EcdsaSignContext_t::ecdsaSignIntBuff**

Internal buffer.

2.21.30.3.3 mbedtls_md_context_t **EcdsaSignContext_t::hash_ctx**

The hash context.

2.21.30.3.4 CCEcpkiHashOpMode_t **EcdsaSignContext_t::hashMode**

The hash mode.

2.21.30.3.5 CCHashResultBuf_t **EcdsaSignContext_t::hashResult**

The hash result buffer.

2.21.30.3.6 uint32_t EcdsaSignContext_t::hashResultSizeWords

The size of the hash result in words.

The documentation for this struct was generated from the following file:

- **cc_ecpki_types.h**

2.21.31 EcdsaVerifyContext_t struct reference

```
#include <cc_ecpki_types.h>
```

2.21.31.1 Data Fields

- **CCEcpkiUserPublKey_t ECDSA_SignerPublKey**
- **mbedtls_md_context_t hash_ctx**
- **CCHashResultBuf_t hashResult**
- uint32_t **hashResultSizeWords**
- **CCEcpkiHashOpMode_t hashMode**
- **CCEcdsaVerifyIntBuff_t ccEcdsaVerIntBuff**

2.21.31.2 Detailed Description

The context definition for verification operation.

2.21.31.3 Field Documentation

2.21.31.3.1 CCEcdsaVerifyIntBuff_t **EcdsaVerifyContext_t::ccEcdsaVerIntBuff**

Internal buffer.

2.21.31.3.2 CCEcpkiUserPublKey_t **EcdsaVerifyContext_t::ECDSA_SignerPublKey**

The data of the public key.

2.21.31.3.3 mbedtls_md_context_t **EcdsaVerifyContext_t::hash_ctx**

The hash context.

2.21.31.3.4 CCEcpkiHashOpMode_t **EcdsaVerifyContext_t::hashMode**

The hash mode.

2.21.31.3.5 CCHashResultBuf_t **EcdsaVerifyContext_t::hashResult**

The hash result.

2.21.31.3.6 uint32_t EcdsaVerifyContext_t::hashResultSizeWords

The size of the hash result in words.

The documentation for this struct was generated from the following file:

- **cc_ecpki_types.h**

2.21.32 HmacHash_t struct reference

```
#include <cc_general_defs.h>
```

2.21.32.1 Data Fields

- uint16_t **hashResultSize**
- **CCHashOperationMode_t hashMode**

2.21.32.2 Detailed Description

Hash parameters for HMAC operation.

2.21.32.3 Field Documentation

2.21.32.3.1 CCHashOperationMode_t HmacHash_t::hashMode

The hash operation mode.

2.21.32.3.2 uint16_t HmacHash_t::hashResultSize

The size of the HMAC hash result.

The documentation for this struct was generated from the following file:

- o **cc_general_defs.h**

2.21.33 mbedtls_aes_context struct reference

The AES context-type definition.

```
#include <aes.h>
```

2.21.33.1 Data Fields

- int **nr**
- uint32_t ***rk**
- uint32_t **buf** [68]

2.21.33.2 Field Documentation

2.21.33.2.1 uint32_t mbedtls_aes_context::buf[68]

Unaligned data buffer. This buffer can hold 32 extra bytes, which can be used for alignment if VIA padlock is used, or simplifying key expansion in the 256-bit case by generating an extra round key.

2.21.33.2.2 int mbedtls_aes_context::nr

The number of rounds.

2.21.33.2.3 uint32_t*mbedtls_aes_context::rk

AES round keys.

The documentation for this struct was generated from the following file:

- aes.h

2.21.34 mbedtls_ccm_context struct reference

The CCM context-type definition. The CCM context is passed to the APIs called.

```
#include <ccm.h>
```

2.21.34.1 Data Fields

- **mbedtls_cipher_context_t cipher_ctx**

2.21.34.2 Field Documentation

2.21.34.2.1 mbedtls_cipher_context_t **mbedtls_ccm_context::cipher_ctx**

The cipher context used.

The documentation for this struct was generated from the following file:

- ccm.h

2.21.35 mbedtls_chacha20_context struct reference

2.21.35.1 Data Fields

- uint32_t state [16]
- uint8_t **keystream8** [64]
- size_t **keystream_bytes_used**

2.21.35.2 Field Documentation

2.21.35.2.1 uint8_t mbedtls_chacha20_context::keystream8[64]

The state (before round operations).

2.21.35.2.2 size_t mbedtls_chacha20_context::keystream_bytes_used

Leftover keystream bytes.

The documentation for this struct was generated from the following file:

- chacha20.h

2.21.36 mbedtls_chachapoly_context struct reference

2.21.36.1 Data Fields

- **mbedtls_chacha20_context chacha20_ctx**
- **mbedtls_poly1305_context poly1305_ctx**
- uint64_t **aad_len**
- uint64_t **ciphertext_len**
- int **state**
- **mbedtls_chachapoly_mode_t mode**

2.21.36.2 Field Documentation

2.21.36.2.1 uint64_t mbedtls_chachapoly_context::aad_len

The length (bytes) of the Additional Authenticated Data.

2.21.36.2.2 mbedtls_chacha20_context mbedtls_chachapoly_context::chacha20_ctx

The ChaCha20 context.

2.21.36.2.3 uint64_t mbedtls_chachapoly_context::ciphertext_len

The length (bytes) of the ciphertext.

2.21.36.2.4 mbedtls_chachapoly_mode_t mbedtls_chachapoly_context::mode

Cipher mode (encrypt or decrypt).

2.21.36.2.5 mbedtls_poly1305_context mbedtls_chachapoly_context::poly1305_ctx

The Poly1305 context.

2.21.36.2.6 int mbedtls_chachapoly_context::state

The current state of the context.

The documentation for this struct was generated from the following file:

- chachapoly.h

2.21.37 mbedtls_cipher_context_t struct reference

```
#include <cipher.h>
```

2.21.37.1 Data Fields

- const **mbedtls_cipher_info_t** ***cipher_info**
- int **key_bitlen**
- **mbedtls_operation_t operation**
- unsigned char **unprocessed_data** [**MBEDTLS_MAX_BLOCK_LENGTH**]
- size_t **unprocessed_len**
- unsigned char **iv** [**MBEDTLS_MAX_IV_LENGTH**]
- size_t **iv_size**
- void ***cipher_ctx**

2.21.37.2 Detailed Description

Generic cipher context.

2.21.37.3 Field Documentation

2.21.37.3.1 void*mbedtls_cipher_context_t::cipher_ctx

The cipher-specific context.

2.21.37.3.2 const mbedtls_cipher_info_t*mbedtls_cipher_context_t::cipher_info

Information about the associated cipher.

2.21.37.3.3 unsigned char mbedtls_cipher_context_t::iv[MBEDTLS_MAX_IV_LENGTH]

Current IV or NONCE_COUNTER for CTR-mode, data unit (or sector) number for XTS-mode.

2.21.37.3.4 size_t mbedtls_cipher_context_t::iv_size

IV size in bytes, for ciphers with variable-length IVs.

2.21.37.3.5 int mbedtls_cipher_context_t::key_bitlen

Key length to use.

2.21.37.3.6 mbedtls_operation_t **mbedtls_cipher_context_t::operation**

Operation that the key of the context has been initialized for.

2.21.37.3.7 unsigned char mbedtls_cipher_context_t::unprocessed_data[MBEDTLS_MAX_BLOCK_LENGTH]

Buffer for input that has not been processed yet.

2.21.37.3.8 size_t mbedtls_cipher_context_t::unprocessed_len

Number of bytes that have not been processed yet.

The documentation for this struct was generated from the following file:

- cipher.h

2.21.38 mbedtls_cipher_info_t struct reference

```
#include <cipher.h>
```

2.21.38.1 Data Fields

- **mbedtls_cipher_type_t type**
- **mbedtls_cipher_mode_t mode**
- unsigned int **key_bitlen**
- const char ***name**
- unsigned int **iv_size**
- int **flags**
- unsigned int **block_size**
- const **mbedtls_cipher_base_t** ***base**

2.21.38.2 Detailed Description

Cipher information. Allows calling cipher functions in a generic way.

2.21.38.3 Field Documentation

2.21.38.3.1 const mbedtls_cipher_base_t***mbedtls_cipher_info_t::base**

Struct for base cipher information and functions.

2.21.38.3.2 unsigned int mbedtls_cipher_info_t::block_size

The block size, in bytes.

2.21.38.3.3 int mbedtls_cipher_info_t::flags

Bitflag comprised of MBEDTLS_CIPHER_VARIABLE_IV_LEN and MBEDTLS_CIPHER_VARIABLE_KEY_LEN indicating whether the cipher supports variable IV or variable key sizes, respectively.

2.21.38.3.4 unsigned int mbedtls_cipher_info_t::iv_size

IV or nonce size, in bytes. For ciphers that accept variable IV sizes, this is the recommended size.

2.21.38.3.5 unsigned int mbedtls_cipher_info_t::key_bitlen

The cipher key length, in bits. This is the default length for variable sized ciphers. Includes parity bits for ciphers like DES.

2.21.38.3.6 mbedtls_cipher_mode_t mbedtls_cipher_info_t::mode

The cipher mode. For example, MBEDTLS_MODE_CBC.

2.21.38.3.7 const char*mbedtls_cipher_info_t::name

Name of the cipher.

2.21.38.3.8 mbedtls_cipher_type_t mbedtls_cipher_info_t::type

Full cipher identifier. For example, MBEDTLS_CIPHER_AES_256_CBC.

The documentation for this struct was generated from the following file:

- cipher.h

2.21.39 mbedtls_cmac_context_t struct reference

```
#include <cmac.h>
```

2.21.39.1 Data Fields

- unsigned char **state** [**MBEDTLS_CIPHER_BLKSIZE_MAX**]
- unsigned char **unprocessed_block** [**MBEDTLS_CIPHER_BLKSIZE_MAX**]
- size_t **unprocessed_len**

2.21.39.2 Detailed Description

The CMAC context structure.

2.21.39.3 Field Documentation

2.21.39.3.1 unsigned char mbedtls_cmac_context_t::state[MBEDTLS_CIPHER_BLKSIZE_MAX]

The internal state of the CMAC algorithm.

2.21.39.3.2 unsigned char mbedtls_cmac_context_t::unprocessed_block[MBEDTLS_CIPHER_BLKSIZE_MAX]

Unprocessed data - either data that was not block aligned and is still pending processing, or the final block.

2.21.39.3.3 size_t mbedtls_cmac_context_t::unprocessed_len

The length of data pending processing.

The documentation for this struct was generated from the following file:

- o cmac.h

2.21.40 mbedtls_ctr_drbg_context struct reference

The CTR_DRBG context structure.

```
#include <ctr_drbg.h>
```

2.21.40.1 Data Fields

- unsigned char **counter** [16]
- int **reseed_counter**
- int **prediction_resistance**
- size_t **entropy_len**
- int **reseed_interval**
- **mbedtls_aes_context aes_ctx**
- int(***f_entropy**)(void *, unsigned char *, size_t)
- void ***p_entropy**

2.21.40.2 Field Documentation

2.21.40.2.1 mbedtls_aes_context mbedtls_ctr_drbg_context::aes_ctx

The AES context.

2.21.40.2.2 unsigned char mbedtls_ctr_drbg_context::counter[16]

The counter (V).

2.21.40.2.3 size_t mbedtls_ctr_drbg_context::entropy_len

The amount of entropy grabbed on each seed or reseed operation.

2.21.40.2.4 int(*mbedtls_ctr_drbg_context::f_entropy) (void *, unsigned char *, size_t)

The entropy callback function.

2.21.40.2.5 void*mbedtls_ctr_drbg_context::p_entropy

The context for the entropy function.

2.21.40.2.6 int mbedtls_ctr_drbg_context::prediction_resistance

This determines whether prediction resistance is enabled, that is whether to systematically reseed before each random generation.

2.21.40.2.7 int mbedtls_ctr_drbg_context::reseed_counter

The reseed counter.

2.21.40.2.8 int mbedtls_ctr_drbg_context::reseed_interval

The reseed interval.

The documentation for this struct was generated from the following file:

- ctr_drbg.h

2.21.41 mbedtls_dhm_context struct reference

The DHM context structure.

```
#include <dhm.h>
```

2.21.41.1 Data Fields

- size_t **len**
- mbedtls_mpi **P**
- mbedtls_mpi **G**
- mbedtls_mpi **X**
- mbedtls_mpi **GX**
- mbedtls_mpi **GY**

- mbedtls_mpi **K**
- mbedtls_mpi **RP**
- mbedtls_mpi **Vi**
- mbedtls_mpi **Vf**
- mbedtls_mpi **pX**

2.21.41.2 Field Documentation

2.21.41.2.1 mbedtls_mpi mbedtls_dhm_context::G

The generator.

2.21.41.2.2 mbedtls_mpi mbedtls_dhm_context::GX

Our public key = G^X mod P.

2.21.41.2.3 mbedtls_mpi mbedtls_dhm_context::GY

The public key of the peer = G^Y mod P.

2.21.41.2.4 mbedtls_mpi mbedtls_dhm_context::K

The shared secret = G^ (XY) mod P.

2.21.41.2.5 size_t mbedtls_dhm_context::len

The size of P in bytes.

2.21.41.2.6 mbedtls_mpi mbedtls_dhm_context::P

The prime modulus.

2.21.41.2.7 mbedtls_mpi mbedtls_dhm_context::pX

The previous X.

2.21.41.2.8 mbedtls_mpi mbedtls_dhm_context::RP

The cached value = R^2 mod P.

2.21.41.2.9 mbedtls_mpi mbedtls_dhm_context::Vf

The unblinding value.

2.21.41.2.10 mbedtls_mpi mbedtls_dhm_context::Vi

The blinding value.

2.21.41.2.11 mbedtls_mpi mbedtls_dhm_context::X

Our secret value.

The documentation for this struct was generated from the following file:

- dhm.h

2.21.42 mbedtls_ecdh_context struct reference

The ECDH context structure.

```
#include <ecdh.h>
```

2.21.42.1 Data Fields

- **mbedtls_ecp_group grp**
- mbedtls_mpi **d**
- **mbedtls_ecp_point Q**
- **mbedtls_ecp_point Qp**
- mbedtls_mpi **z**
- int **point_format**
- **mbedtls_ecp_point Vi**
- **mbedtls_ecp_point Vf**
- mbedtls_mpi **_d**

2.21.42.2 Detailed Description

Performing multiple operations concurrently on the same ECDSA context is not supported; objects of this type should not be shared between multiple threads.

2.21.42.3 Field Documentation

2.21.42.3.1 mbedtls_mpi mbedtls_ecdh_context::_d

The previous d.

2.21.42.3.2 mbedtls_mpi mbedtls_ecdh_context::d

The private key.

2.21.42.3.3 mbedtls_ecp_group **mbedtls_ecdh_context::grp**

The elliptic curve used.

2.21.42.3.4 int mbedtls_ecdh_context::point_format

The format of point export in TLS messages.

2.21.42.3.5 mbedtls_ecp_point **mbedtls_ecdh_context::Q**

The public key.

2.21.42.3.6 mbedtls_ecp_point **mbedtls_ecdh_context::Qp**

The value of the public key of the peer.

2.21.42.3.7 mbedtls_ecp_point **mbedtls_ecdh_context::Vf**

The unblinding value.

2.21.42.3.8 mbedtls_ecp_point **mbedtls_ecdh_context::Vi**

The blinding value.

2.21.42.3.9 mbedtls_mpi mbedtls_ecdh_context::z

The shared secret.

The documentation for this struct was generated from the following file:

- o **ecdh.h**

2.21.43 mbedtls_ecp_curve_info struct reference

```
#include <ecp.h>
```

2.21.43.1 Data Fields

- **mbedtls_ecp_group_id grp_id**
- uint16_t **tls_id**
- uint16_t **bit_size**
- const char ***name**

2.21.43.2 Detailed Description

Curve information, for use by other modules.

2.21.43.3 Field Documentation

2.21.43.3.1 uint16_t mbedtls_ecp_curve_info::bit_size

The curve size in bits.

2.21.43.3.2 mbedtls_ecp_group_id mbedtls_ecp_curve_info::grp_id

An internal identifier.

2.21.43.3.3 const char*mbedtls_ecp_curve_info::name

A human-friendly name.

2.21.43.3.4 uint16_t mbedtls_ecp_curve_info::tls_id

The TLS NamedCurve identifier.

The documentation for this struct was generated from the following file:

- ecp.h

2.21.44 mbedtls_ecp_group struct reference

The ECP group structure.

```
#include <ecp.h>
```

2.21.44.1 Data Fields

- **mbedtls_ecp_group_id id**
- mbedtls_mpi **P**
- mbedtls_mpi **A**
- mbedtls_mpi **B**
- **mbedtls_ecp_point G**
- mbedtls_mpi **N**
- size_t **pbits**
- size_t **nbits**
- unsigned int h
- int(***modp**)(mbedtls_mpi *)

- int(***t_pre**)(**mbedtls_ecp_point** *, void *)
- int(***t_post**)(**mbedtls_ecp_point** *, void *)
- void ***t_data**
- **mbedtls_ecp_point** ***T**
- size_t **T_size**

2.21.44.2 Detailed Description

We consider two types of curve equations:

- Short Weierstrass: y^2 = x^3 + A x + B mod P (*SECG SEC1* + *RFC-4492*)
- Montgomery: y^2 = x^3 + A x^2 + x mod P (*Curve25519*, *Curve448*)

In both cases, the generator (G) for a prime-order subgroup is fixed.

For Short Weierstrass, this subgroup is the whole curve, and its cardinality is denoted by N. Our code requires that N is an odd prime as **mbedtls_ecp_mul()** requires an odd number, and **mbedtls_ecdsa_sign()** requires that it is prime for blinding purposes.

For Montgomery curves, we do not store A , but (A + 2) / 4 , which is the quantity used in the formulas. Additionally, nbits is not the size of N but the required size for private keys.

If modp is NULL, reduction modulo P is done using a generic algorithm. Otherwise, modp must point to a function that takes an mbedtls_mpi in the range of 0..2^(2*pbits)-1 , and transforms it in-place to an integer which is congruent mod P to the given MPI, and is close enough to pbits in size, so that it may be efficiently brought in the 0..P-1 range by a few additions or subtractions. Therefore, it is only an approximative modular reduction. It must return 0 on success and non-zero on failure.

Alternative implementations must keep the group IDs distinct. If two group structures have the same ID, then they must be identical.

2.21.44.3 Field Documentation

2.21.44.3.1 mbedtls_mpi mbedtls_ecp_group::A

For Short Weierstrass: A in the equation. For Montgomery curves: (A + 2) / 4.

2.21.44.3.2 mbedtls_mpi mbedtls_ecp_group::B

For Short Weierstrass: B in the equation. For Montgomery curves: unused.

2.21.44.3.3 mbedtls_ecp_point mbedtls_ecp_group::G

The generator of the subgroup used.

2.21.44.3.4 mbedtls_ecp_group_id **mbedtls_ecp_group::id**

An internal group identifier.

2.21.44.3.5 int(*mbedtls_ecp_group::modp) (mbedtls_mpi *)

The function for fast pseudo-reduction mod P (see above).

2.21.44.3.6 mbedtls_mpi mbedtls_ecp_group::N

The order of G.

2.21.44.3.7 size_t mbedtls_ecp_group::nbits

For Short Weierstrass: The number of bits in P. For Montgomery curves: the number of bits in the private keys.

2.21.44.3.8 mbedtls_mpi mbedtls_ecp_group::P

The prime modulus of the base field.

2.21.44.3.9 size_t mbedtls_ecp_group::pbits

The number of bits in P.

2.21.44.3.10 mbedtls_ecp_point***mbedtls_ecp_group::T**

Pre-computed points for ecp_mul_comb().

2.21.44.3.11 void*mbedtls_ecp_group::t_data

Unused.

2.21.44.3.12 int(*mbedtls_ecp_group::t_post) (mbedtls_ecp_point *, void *)

Unused.

2.21.44.3.13 int(*mbedtls_ecp_group::t_pre) (mbedtls_ecp_point *, void *)

Unused.

2.21.44.3.14 size_t mbedtls_ecp_group::T_size

The number of pre-computed points.

The documentation for this struct was generated from the following file:

- ecp.h

2.21.45 mbedtls_ecp_keypair struct reference

The ECP key-pair structure.

```
#include <ecp.h>
```

2.21.45.1 Data Fields

- **mbedtls_ecp_group grp**
- mbedtls_mpi **d**
- **mbedtls_ecp_point Q**

2.21.45.2 Detailed Description

A generic key-pair that may be used for ECDSA and fixed ECDH, for example.

Members are deliberately in the same order as in the **mbedtls_ecdsa_context** structure.

2.21.45.3 Field Documentation

2.21.45.3.1 mbedtls_mpi mbedtls_ecp_keypair::d

our secret value

2.21.45.3.2 mbedtls_ecp_group mbedtls_ecp_keypair::grp

Elliptic curve and base point

2.21.45.3.3 mbedtls_ecp_point mbedtls_ecp_keypair::Q

our public value

The documentation for this struct was generated from the following file:

- ecp.h

2.21.46 mbedtls_ecp_point struct reference

The ECP point structure, in Jacobian coordinates.

```
#include <ecp.h>
```

2.21.46.1 Data Fields

- mbedtls_mpi **X**

- mbedtls_mpi **Y**
- mbedtls_mpi **Z**

2.21.46.2 Detailed Description

All functions expect and return points satisfying the following condition: Z == 0 or Z == 1. Other values of Z are used only by internal functions. The point is zero, or "at infinity", if Z == 0. Otherwise, X and Y are its standard (affine) coordinates.

2.21.46.3 Field Documentation

2.21.46.3.1 mbedtls_mpi mbedtls_ecp_point::X

The X coordinate of the ECP point.

2.21.46.3.2 mbedtls_mpi mbedtls_ecp_point::Y

The Y coordinate of the ECP point.

2.21.46.3.3 mbedtls_mpi mbedtls_ecp_point::Z

The Z coordinate of the ECP point.

The documentation for this struct was generated from the following file:

- ecp.h

2.21.47 mbedtls_gcm_context struct reference

The GCM context structure.

```
#include <gcm.h>
```

2.21.47.1 Data Fields

- **mbedtls_cipher_context_t cipher_ctx**
- uint64_t **HL** [16]
- uint64_t **HH** [16]
- uint64_t **len**
- uint64_t **add_len**
- unsigned char **base_ectr** [16]
- unsigned char **y** [16]

- unsigned char **buf** [16]
- int **mode**

2.21.47.2 Field Documentation

2.21.47.2.1 uint64_t mbedtls_gcm_context::add_len

The total length of the additional data.

2.21.47.2.2 unsigned char mbedtls_gcm_context::base_ectr[16]

The first ECTR for tag.

2.21.47.2.3 unsigned char mbedtls_gcm_context::buf[16]

The buf working value.

2.21.47.2.4 mbedtls_cipher_context_t mbedtls_gcm_context::cipher_ctx

The cipher context used.

2.21.47.2.5 uint64_t mbedtls_gcm_context::HH[16]

Precalculated HTable high.

2.21.47.2.6 uint64_t mbedtls_gcm_context::HL[16]

Precalculated HTable low.

2.21.47.2.7 uint64_t mbedtls_gcm_context::len

The total length of the encrypted data.

2.21.47.2.8 int mbedtls_gcm_context::mode

The operation to perform: #MBEDTLS_GCM_ENCRYPT or #MBEDTLS_GCM_DECRYPT.

2.21.47.2.9 unsigned char mbedtls_gcm_context::y[16]

The Y working value.

The documentation for this struct was generated from the following file:

- gcm.h

2.21.48 mbedtls_md_context_t struct reference

```
#include <md.h>
```

2.21.48.1 Data Fields

- const **mbedtls_md_info_t** ***md_info**
- void ***md_ctx**
- void ***hmac_ctx**

2.21.48.2 Detailed Description

The generic message-digest context.

2.21.48.3 Field Documentation

2.21.48.3.1 void*mbedtls_md_context_t::hmac_ctx

The HMAC part of the context.

2.21.48.3.2 void*mbedtls_md_context_t::md_ctx

The digest-specific context.

2.21.48.3.3 const mbedtls_md_info_t*mbedtls_md_context_t::md_info

Information about the associated message digest.

The documentation for this struct was generated from the following file:

- o **md.h**

2.21.49 mbedtls_mng_apbc_part union reference

```
#include <mbedtls_cc_mng.h>
```

2.21.49.1 Data Fields

- uint8_t **apbcPartVal**
- struct {

  uint8_t **accessAllow**: 1

  uint8_t **accessAllowLock**: 1

  uint8_t **accessModify**: 1

  uint8_t **rfu**: 5

} **apbcPartBits**

2.21.49.2 Detailed Description

A uint8_t representation for the APB-C parts in the AO_APB_FILTERING register.

2.21.49.3 Field Documentation

2.21.49.3.1 uint8_t mbedtls_mng_apbc_part::accessAllow

APB-C accepts only 'mbedtls_mng_apbc_parts' accesses.

2.21.49.3.2 uint8_t mbedtls_mng_apbc_part::accessAllowLock

APB-C accessAllow cannot be modified.

2.21.49.3.3 uint8_t mbedtls_mng_apbc_part::accessModify

User decided to modify the upper couple.

2.21.49.3.4 struct { ... } mbedtls_mng_apbc_part::apbcPartBits

A representation of the APB-C parts in the AO_APB_FILTERING register.

2.21.49.3.5 uint8_t mbedtls_mng_apbc_part::apbcPartVal

A representation of the APB-C value in the AO_APB_FILTERING register.

2.21.49.3.6 uint8_t mbedtls_mng_apbc_part::rfu

APB-C part access bits.

The documentation for this union was generated from the following file:

- o **mbedtls_cc_mng.h**

2.21.50 mbedtls_mng_apbcconfig union reference

```
#include <mbedtls_cc_mng.h>
```

2.21.50.1 Data Fields

- uint32_t **apbcConfigVal**
- **mbedtls_mng_apbc_part apbcPart** [**CC_MNG_APBC_TOTAL_ID**+1]

2.21.50.2 Detailed Description

Input to the **mbedtls_mng_apbc_config_set()** function.

2.21.50.3 Field Documentation

2.21.50.3.1 uint32_t mbedtls_mng_apbcconfig::apbcConfigVal

APB-C configuration values.

2.21.50.3.2 mbedtls_mng_apbc_part mbedtls_mng_apbcconfig::apbcPart[CC_MNG_APBC_TOTAL_ID+1]

An array of the configuration bits for the Secure, Privileged, and Instruction parts.

The documentation for this union was generated from the following file:

- **mbedtls_cc_mng.h**

2.21.51 mbedtls_nist_kw_context struct reference

The key wrapping context-type definition. The key wrapping context is passed to the APIs called.

```
#include <nist_kw.h>
```

2.21.51.1 Data Fields

- **mbedtls_cipher_context_t cipher_ctx**

2.21.51.2 Detailed Description

The definition of this type may change in future library versions. Don't make any assumptions on this context.

2.21.51.3 Field Documentation

2.21.51.3.1 mbedtls_cipher_context_t mbedtls_nist_kw_context::cipher_ctx

The cipher context used.

The documentation for this struct was generated from the following file:

- **nist_kw.h**

2.21.52 mbedtls_platform_context struct reference

The platform context structure.

```
#include <platform.h>
```

2.21.52.1 Data Fields

- char **dummy**

2.21.52.2 Detailed Description

This structure may be used to assist platform-specific setup or teardown operations.

2.21.52.3 Field Documentation

2.21.52.3.1 char mbedtls_platform_context::dummy

A placeholder member, as empty structs are not portable.

The documentation for this struct was generated from the following file:

- platform.h

2.21.53 mbedtls_poly1305_context struct reference

2.21.53.1 Data Fields

- uint32_t r [4]
- uint32_t **s** [4]
- uint32_t **acc** [5]
- uint8_t **queue** [16]
- size_t **queue_len**

2.21.53.2 Field Documentation

2.21.53.2.1 uint32_t mbedtls_poly1305_context::acc[5]

The value for 's' (high 128 bits of the key).

2.21.53.2.2 uint8_t mbedtls_poly1305_context::queue[16]

The accumulator number.

2.21.53.2.3 size_t mbedtls_poly1305_context::queue_len

The current partial block of data.

2.21.53.2.4 uint32_t mbedtls_poly1305_context::s[4]

The value for 'r' (low 128 bits of the key).

The documentation for this struct was generated from the following file:

- poly1305.h

2.21.54 mbedtls_rsa_context struct reference

The RSA context structure.

```
#include <rsa.h>
```

2.21.54.1 Data Fields

- int **ver**
- size_t **len**
- mbedtls_mpi **N**
- mbedtls_mpi **E**
- mbedtls_mpi **D**
- mbedtls_mpi **P**
- mbedtls_mpi **Q**
- mbedtls_mpi **DP**
- mbedtls_mpi **DQ**
- mbedtls_mpi **QP**
- mbedtls_mpi **RN**
- mbedtls_mpi **RP**
- mbedtls_mpi **RQ**
- mbedtls_mpi **Vi**
- mbedtls_mpi **Vf**
- int **padding**
- int **hash_id**

2.21.54.2 Detailed Description

Direct manipulation of the members of this structure is deprecated. All manipulation should instead be done through the public interface functions.

2.21.54.3 Field Documentation

2.21.54.3.1 mbedtls_mpi mbedtls_rsa_context::D

The private exponent.

2.21.54.3.2 mbedtls_mpi mbedtls_rsa_context::DP

D % (P - 1).

2.21.54.3.3 mbedtls_mpi mbedtls_rsa_context::DQ

D % (Q - 1).

2.21.54.3.4 mbedtls_mpi mbedtls_rsa_context::E

The public exponent.

2.21.54.3.5 int mbedtls_rsa_context::hash_id

Hash identifier of mbedtls_md_type_t type, as specified in **md.h** for use in the MGF mask generating function used in the EME-OAEP and EMSA-PSS encodings.

2.21.54.3.6 size_t mbedtls_rsa_context::len

The size of N in bytes.

2.21.54.3.7 mbedtls_mpi mbedtls_rsa_context::N

The public modulus.

2.21.54.3.8 mbedtls_mpi mbedtls_rsa_context::P

The first prime factor.

2.21.54.3.9 int mbedtls_rsa_context::padding

Selects padding mode: **MBEDTLS_RSA_PKCS_V15** for 1.5 padding and **MBEDTLS_RSA_PKCS_V21** for OAEP or PSS.

2.21.54.3.10 mbedtls_mpi mbedtls_rsa_context::Q

The second prime factor.

2.21.54.3.11 mbedtls_mpi mbedtls_rsa_context::QP

1 / (Q % P).

2.21.54.3.12 mbedtls_mpi mbedtls_rsa_context::RN

cached R^2 mod N.

2.21.54.3.13 mbedtls_mpi mbedtls_rsa_context::RP

cached R^2 mod P.

2.21.54.3.14 mbedtls_mpi mbedtls_rsa_context::RQ

cached R^2 mod Q.

2.21.54.3.15 int mbedtls_rsa_context::ver

Always 0.

2.21.54.3.16 mbedtls_mpi mbedtls_rsa_context::Vf

The cached un-blinding value.

2.21.54.3.17 mbedtls_mpi mbedtls_rsa_context::Vi

The cached blinding value.

The documentation for this struct was generated from the following file:

- rsa.h

2.21.55 mbedtls_sha1_context struct reference

The SHA-1 context structure.

```
#include <sha1.h>
```

2.21.55.1 Data Fields

- uint32_t **total** [2]
- uint32_t **state** [5]
- unsigned char **buffer** [64]

2.21.55.2 Detailed Description

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

2.21.55.3 Field Documentation

2.21.55.3.1 unsigned char mbedtls_sha1_context::buffer[64]

The data block being processed.

2.21.55.3.2 uint32_t mbedtls_sha1_context::state[5]

The intermediate digest state.

2.21.55.3.3 uint32_t mbedtls_sha1_context::total[2]

The number of bytes processed.

The documentation for this struct was generated from the following file:

- sha1.h

2.21.56 mbedtls_sha256_context struct reference

The SHA-256 context structure.

```
#include <sha256.h>
```

2.21.56.1 Data Fields

- uint32_t **total** [2]
- uint32_t **state** [8]
- unsigned char **buffer** [64]
- int **is224**

2.21.56.2 Detailed Description

The structure is used both for SHA-256 and for SHA-224 checksum calculations. The choice between these two is made in the call to **mbedtls_sha256_starts_ret()**.

2.21.56.3 Field Documentation

2.21.56.3.1 unsigned char mbedtls_sha256_context::buffer[64]

The data block being processed.

2.21.56.3.2 int mbedtls_sha256_context::is224

Determines which function to use: 0: Use SHA-256, or 1: Use SHA-224.

2.21.56.3.3 uint32_t mbedtls_sha256_context::state[8]

The intermediate digest state.

2.21.56.3.4 uint32_t mbedtls_sha256_context::total[2]

The number of bytes processed.

The documentation for this struct was generated from the following file:

- sha256.h

2.21.57 mbedtls_sha512_context struct reference

The SHA-512 context structure.

```
#include <sha512.h>
```

2.21.57.1 Data Fields

- uint64_t **total** [2]
- uint64_t **state** [8]
- unsigned char **buffer** [128]
- int **is384**

2.21.57.2 Detailed Description

The structure is used both for SHA-384 and for SHA-512 checksum calculations. The choice between these two is made in the call to **mbedtls_sha512_starts_ret()**.

2.21.57.3 Field Documentation

2.21.57.3.1 unsigned char mbedtls_sha512_context::buffer[128]

The data block being processed.

2.21.57.3.2 int mbedtls_sha512_context::is384

Determines which function to use: 0: Use SHA-512, or 1: Use SHA-384.

2.21.57.3.3 uint64_t mbedtls_sha512_context::state[8]

The intermediate digest state.

2.21.57.3.4 uint64_t mbedtls_sha512_context::total[2]

The number of bytes processed.

The documentation for this struct was generated from the following file:

- sha512.h

2.21.58 mbedtls_srp_context struct reference

```
#include <mbedtls_cc_srp.h>
```

2.21.58.1 Data Fields

- **mbedtls_srp_entity_t srpType**
- **mbedtls_srp_version_t srpVer**
- **mbedtls_srp_group_param groupParam**
- **CCHashOperationMode_t hashMode**
- size_t **hashDigestSize**
- size_t **sessionKeySize**
- **CCRndContext_t** *__pRndCtx__
- **mbedtls_srp_modulus ephemPriv**
- size_t **ephemPrivSize**
- **mbedtls_srp_digest userNameDigest**
- **mbedtls_srp_digest credDigest**
- **mbedtls_srp_digest kMult**

2.21.58.2 Detailed Description

The SRP context prototype

2.21.58.3 Field Documentation

2.21.58.3.1 mbedtls_srp_digest **mbedtls_srp_context::credDigest**

The cred digest.

2.21.58.3.2 mbedtls_srp_modulus **mbedtls_srp_context::ephemPriv**

The modulus.

2.21.58.3.3 size_t mbedtls_srp_context::ephemPrivSize

The modulus size.

2.21.58.3.4 mbedtls_srp_group_param **mbedtls_srp_context::groupParam**

The group parameter including the modulus information.

2.21.58.3.5 size_t mbedtls_srp_context::hashDigestSize

The hash digest size.

2.21.58.3.6 CCHashOperationMode_t **mbedtls_srp_context::hashMode**

The hash mode.

2.21.58.3.7 mbedtls_srp_digest **mbedtls_srp_context::kMult**

The SRP K multiplier.

2.21.58.3.8 CCRndContext_t* **mbedtls_srp_context::pRndCtx**

A pointer to the RND context.

2.21.58.3.9 size_t mbedtls_srp_context::sessionKeySize

The session key size.

2.21.58.3.10 mbedtls_srp_entity_t **mbedtls_srp_context::srpType**

The SRP entitiy type.

2.21.58.3.11 mbedtls_srp_version_t **mbedtls_srp_context::srpVer**

The SRP version.

2.21.58.3.12 mbedtls_srp_digest **mbedtls_srp_context::userNameDigest**

The user-name digest.

The documentation for this struct was generated from the following file:

- **mbedtls_cc_srp.h**

2.21.59 mbedtls_srp_group_param struct reference

Group parameters for the SRP.

```
#include <mbedtls_cc_srp.h>
```

2.21.59.1 Data Fields

- **mbedtls_srp_modulus modulus**
- uint8_t **gen**
- size_t **modSizeInBits**
- uint32_t **validNp**
- uint32_t **Np** [**CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS**]

2.21.59.2 Detailed Description

Defines the modulus and the generator used.

2.21.59.3 Field Documentation

2.21.59.3.1 uint8_t mbedtls_srp_group_param::gen

The SRP generator.

2.21.59.3.2 size_t mbedtls_srp_group_param::modSizeInBits

The size of the SRP modulus in bits.

2.21.59.3.3 mbedtls_srp_modulus **mbedtls_srp_group_param::modulus**

The SRP modulus.

2.21.59.3.4 uint32_t mbedtls_srp_group_param::Np[CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS]

The SRP Np buffer.

2.21.59.3.5 uint32_t mbedtls_srp_group_param::validNp

The valid SRP Np.

The documentation for this struct was generated from the following file:

- **mbedtls_cc_srp.h**

2.21.60 mbedtls_util_keydata struct reference

```
#include <mbedtls_cc_util_defs.h>
```

2.21.60.1 Data Fields

- uint8_t ***pKey**
- size_t **keySize**

2.21.60.2 Detailed Description

The key data.

2.21.60.3 Field Documentation

2.21.60.3.1 size_t mbedtls_util_keydata::keySize

The size of the key in bytes.

2.21.60.3.2 uint8_t*mbedtls_util_keydata::pKey

A pointer to the key.

The documentation for this struct was generated from the following file:

- **mbedtls_cc_util_defs.h**

3 Mbed TLS cryptographic API layer

3.1 Deprecated list

Global mbedtls_aes_decrypt **(**mbedtls_aes_context ***ctx, const unsigned char input[16], unsigned char output[16])**

Superseded by **mbedtls_internal_aes_decrypt()**

Global mbedtls_aes_encrypt **(**mbedtls_aes_context ***ctx, const unsigned char input[16], unsigned char output[16])**

Superseded by **mbedtls_internal_aes_encrypt()**

Global mbedtls_ctr_drbg_update **(**mbedtls_ctr_drbg_context ***ctx, const unsigned char *additional, size_t add_len)**

Superseded by **mbedtls_ctr_drbg_update_ret()** in 2.16.0.

Global MBEDTLS_DHM_RFC3526_MODP_2048_P

The hex-encoded primes from *RFC-3526* are deprecated and superseded by the corresponding macros providing them as binary constants. Their hex-encoded constants are likely to be removed in a future version of the library.

Global MBEDTLS_DHM_RFC5114_MODP_2048_P

The hex-encoded primes from *RFC-5114* are deprecated and are likely to be removed in a future version of the library without replacement.

Global mbedtls_md_init_ctx **(**mbedtls_md_context_t ***ctx, const mbedtls_md_info_t *md_info) MBEDTLS_DEPRECATED**

Superseded by **mbedtls_md_setup()** in 2.0.0

Global mbedtls_rsa_pkcs1_decrypt **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_pkcs1_encrypt **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_rsa_pkcs1_sign **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_pkcs1_verify **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_rsa_rsaes_oaep_decrypt **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_rsaes_oaep_encrypt **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t ilen, const unsigned char *input, unsigned char *output)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_rsa_rsaes_pkcs1_v15_decrypt **(**mbedtls_rsa_context ***ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_rsaes_pkcs1_v15_encrypt (mbedtls_rsa_context *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_rsa_rsassa_pkcs1_v15_sign (mbedtls_rsa_context *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_rsassa_pkcs1_v15_verify (mbedtls_rsa_context *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_rsa_rsassa_pss_sign (mbedtls_rsa_context *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Global mbedtls_rsa_rsassa_pss_verify (mbedtls_rsa_context *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Global mbedtls_sha1 (const unsigned char *input, size_t ilen, unsigned char output[20])

Superseded by **mbedtls_sha1_ret()** in 2.7.0

Global mbedtls_sha1_finish **(**mbedtls_sha1_context ***ctx, unsigned char output[20])**

Superseded by **mbedtls_sha1_finish_ret()** in 2.7.0.

Global mbedtls_sha1_process **(**mbedtls_sha1_context ***ctx, const unsigned char data[64])**

Superseded by **mbedtls_internal_sha1_process()** in 2.7.0.

Global mbedtls_sha1_starts **(**mbedtls_sha1_context ***ctx)**

Superseded by **mbedtls_sha1_starts_ret()** in 2.7.0.

Global mbedtls_sha1_update **(**mbedtls_sha1_context ***ctx, const unsigned char *input, size_t ilen)**

Superseded by **mbedtls_sha1_update_ret()** in 2.7.0.

Global mbedtls_sha256 **(const unsigned char *input, size_t ilen, unsigned char output[32], int is224)**

Superseded by **mbedtls_sha256_ret()** in 2.7.0.

Global mbedtls_sha256_finish **(**mbedtls_sha256_context ***ctx, unsigned char output[32])**

Superseded by **mbedtls_sha256_finish_ret()** in 2.7.0.

Global mbedtls_sha256_process **(**mbedtls_sha256_context ***ctx, const unsigned char data[64])**

Superseded by **mbedtls_internal_sha256_process()** in 2.7.0.

Global mbedtls_sha256_starts **(**mbedtls_sha256_context ***ctx, int is224)**

Superseded by **mbedtls_sha256_starts_ret()** in 2.7.0.

Global mbedtls_sha256_update **(**mbedtls_sha256_context ***ctx, const unsigned char *input, size_t ilen)**

Superseded by **mbedtls_sha256_update_ret()** in 2.7.0.

Global mbedtls_sha512 **(const unsigned char *input, size_t ilen, unsigned char output[64], int is384)**

Superseded by **mbedtls_sha512_ret()** in 2.7.0

Global mbedtls_sha512_finish **(**mbedtls_sha512_context ***ctx, unsigned char output[64])**

Superseded by **mbedtls_sha512_finish_ret()** in 2.7.0.

Global mbedtls_sha512_process **(**mbedtls_sha512_context ***ctx, const unsigned char data[128])**

Superseded by **mbedtls_internal_sha512_process()** in 2.7.0.

Global mbedtls_sha512_starts **(**mbedtls_sha512_context ***ctx, int is384)**

Superseded by **mbedtls_sha512_starts_ret()** in 2.7.0

Global mbedtls_sha512_update **(**mbedtls_sha512_context ***ctx, const unsigned char *input, size_t ilen)**

Superseded by **mbedtls_sha512_update_ret()** in 2.7.0.

3.2 aes.h File reference

This file contains AES definitions and functions.

```
#include "config.h"

#include <stddef.h>

#include <stdint.h>
```

3.2.1 Data structures

- struct **mbedtls_aes_context**

  The AES context-type definition.

3.2.2 Macros

- #define **MBEDTLS_AES_ENCRYPT** 1
- #define **MBEDTLS_AES_DECRYPT** 0
- #define **MBEDTLS_ERR_AES_INVALID_KEY_LENGTH** -0x0020
- #define **MBEDTLS_ERR_AES_INVALID_INPUT_LENGTH** -0x0022
- #define **MBEDTLS_ERR_AES_BAD_INPUT_DATA** -0x0021
- #define **MBEDTLS_ERR_AES_FEATURE_UNAVAILABLE** -0x0023
- #define **MBEDTLS_ERR_AES_HW_ACCEL_FAILED** -0x0025
- #define MBEDTLS_DEPRECATED

3.2.3 typedefs

- typedef struct **mbedtls_aes_context mbedtls_aes_context**

  The AES context-type definition.

3.2.4 Functions

- void **mbedtls_aes_init** (**mbedtls_aes_context** *ctx)

  This function initializes the specified AES context.

- void **mbedtls_aes_free** (**mbedtls_aes_context** *ctx)

  This function releases and clears the specified AES context.

- int **mbedtls_aes_setkey_enc** (**mbedtls_aes_context** *ctx, const unsigned char *key, unsigned int keybits)

This function sets the encryption key.

- int **mbedtls_aes_setkey_dec** (**mbedtls_aes_context** *ctx, const unsigned char *key, unsigned int keybits)

  This function sets the decryption key.

- int **mbedtls_aes_crypt_ecb** (**mbedtls_aes_context** *ctx, int mode, const unsigned char input[16], unsigned char output[16])

  This function performs an AES single-block encryption or decryption operation.

- int **mbedtls_internal_aes_encrypt** (**mbedtls_aes_context** *ctx, const unsigned char input[16], unsigned char output[16])

  Internal AES block encryption function. This is only exposed to allow overriding it using MBEDTLS_AES_ENCRYPT_ALT.

- int **mbedtls_internal_aes_decrypt** (**mbedtls_aes_context** *ctx, const unsigned char input[16], unsigned char output[16])

  Internal AES block decryption function. This is only exposed to allow overriding it using see MBEDTLS_AES_DECRYPT_ALT.

- MBEDTLS_DEPRECATED void **mbedtls_aes_encrypt** (**mbedtls_aes_context** *ctx, const unsigned char input[16], unsigned char output[16])

  Deprecated internal AES block encryption function without return value.

- MBEDTLS_DEPRECATED void **mbedtls_aes_decrypt** (**mbedtls_aes_context** *ctx, const unsigned char input[16], unsigned char output[16])

  Deprecated internal AES block decryption function without return value.

3.2.5 Detailed description

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data.

The AES algorithm is a symmetric block cipher that can encrypt and decrypt information. For more information, see *FIPS 197 Advanced Encryption Standard* and *ISO/IEC 18033-2:2006 Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*.

The AES-XTS block mode is standardized by *NIST SP 800-38E* **https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38e.pdf** and described in detail by *IEEE P1619* **https://ieeexplore.ieee.org/document/4375278**.

3.2.6 Macro definition documentation

3.2.6.1 #define MBEDTLS_AES_DECRYPT 0

AES decryption.

3.2.6.2 #define MBEDTLS_AES_ENCRYPT 1

AES encryption.

3.2.6.3 #define MBEDTLS_ERR_AES_BAD_INPUT_DATA -0x0021

Invalid input data.

3.2.6.4 #define MBEDTLS_ERR_AES_FEATURE_UNAVAILABLE -0x0023

Feature not available. For example, an unsupported AES key size.

3.2.6.5 #define MBEDTLS_ERR_AES_HW_ACCEL_FAILED -0x0025

AES hardware accelerator failed.

3.2.6.6 #define MBEDTLS_ERR_AES_INVALID_INPUT_LENGTH -0x0022

Invalid data input length.

3.2.6.7 #define MBEDTLS_ERR_AES_INVALID_KEY_LENGTH -0x0020

Invalid key length.

3.2.7 Function documentation

3.2.7.1 int mbedtls_aes_crypt_ecb (mbedtls_aes_context * ctx, int mode, const unsigned char input[16], unsigned char output[16])

It performs the operation defined in the mode parameter (encrypt or decrypt), on the input data buffer defined in the input parameter.

mbedtls_aes_init(), and either **mbedtls_aes_setkey_enc()** or **mbedtls_aes_setkey_dec()** must be called before the first call to this API with the same context.

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption or decryption. It must be initialized and bound to a key.
`mode`	The AES operation: **MBEDTLS_AES_ENCRYPT** or **MBEDTLS_AES_DECRYPT**.
`input`	The buffer holding the input data. It must be readable and at least 16 bytes long.
`output`	The buffer where the output data will be written. It must be writeable and at least 16 bytes long.

Returns:

0 on success.

3.2.7.2 MBEDTLS_DEPRECATED void mbedtls_aes_decrypt (mbedtls_aes_context * ctx, const unsigned char input[16], unsigned char output[16])

Deprecated:

Superseded by **mbedtls_internal_aes_decrypt()**

Parameters:

Parameter	Description
`ctx`	The AES context to use for decryption.
`input`	Ciphertext block.
`output`	Output (plaintext) block.

3.2.7.3 MBEDTLS_DEPRECATED void mbedtls_aes_encrypt (mbedtls_aes_context * ctx, const unsigned char input[16], unsigned char output[16])

Deprecated:

Superseded by **mbedtls_internal_aes_encrypt()**

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption.
`input`	Plaintext block.
`output`	Output (ciphertext) block.

3.2.7.4 void mbedtls_aes_free (mbedtls_aes_context * ctx)

Parameters:

Parameter	Description
`ctx`	The AES context to clear. If this is NULL , this function does nothing. Otherwise, the context must have been at least initialized.

3.2.7.5 void mbedtls_aes_init (mbedtls_aes_context * ctx)

It must be the first API called before using the context.

Parameters:

Parameter	Description
`ctx`	The AES context to initialize. This must not be NULL.

3.2.7.6 int mbedtls_aes_setkey_dec (mbedtls_aes_context * ctx, const unsigned char * key, unsigned int keybits)

Parameters:

Parameter	Description
`ctx`	The AES context to which the key should be bound. It must be initialized.
`key`	The decryption key. This must be a readable buffer of size keybits bits.
`keybits`	The size of data passed. Valid options are: 128 bits, 192 bits, or 256 bits.

Returns:

0 on success.

MBEDTLS_ERR_AES_INVALID_KEY_LENGTH on failure.

3.2.7.7 int mbedtls_aes_setkey_enc (mbedtls_aes_context * ctx, const unsigned char * key, unsigned int keybits)

Parameters:

Parameter	Description
`ctx`	The AES context to which the key should be bound. It must be initialized.
`key`	The encryption key. This must be a readable buffer of size keybits bits.
`keybits`	The size of data passed in bits. Valid options are: 128 bits, 192 bits, or 256 bits.

Returns:

0 on success.

MBEDTLS_ERR_AES_INVALID_KEY_LENGTH on failure.

3.2.7.8 int mbedtls_internal_aes_decrypt (mbedtls_aes_context * ctx, const unsigned char input[16], unsigned char output[16])

Parameters:

Parameter	Description
`ctx`	The AES context to use for decryption.
`input`	The ciphertext block.
`output`	The output (plaintext) block.

Returns:

0 on success.

3.2.7.9 int mbedtls_internal_aes_encrypt (mbedtls_aes_context * ctx, const unsigned char input[16], unsigned char output[16])

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption.
`input`	The plaintext block.
`output`	The output (ciphertext) block.

Returns:

0 on success.

3.3 ccm.h File reference

This file provides an API for the CCM authenticated encryption mode for block ciphers.

```
#include "config.h"
```

```
#include "cipher.h"
```

3.3.1 Data structures

- struct **mbedtls_ccm_context**

  The CCM context-type definition. The CCM context is passed to the APIs called.

3.3.2 Macros

- #define **MBEDTLS_ERR_CCM_BAD_INPUT** -0x000D
- #define **MBEDTLS_ERR_CCM_AUTH_FAILED** -0x000F
- #define **MBEDTLS_ERR_CCM_HW_ACCEL_FAILED** -0x0011

3.3.3 typedefs

- typedef struct **mbedtls_ccm_context mbedtls_ccm_context**

  The CCM context-type definition. The CCM context is passed to the APIs called.

3.3.4 Functions

- void **mbedtls_ccm_init** (**mbedtls_ccm_context** *ctx)

  This function initializes the specified CCM context, to make references valid, and prepare the context for **mbedtls_ccm_setkey()** or **mbedtls_ccm_free()**.

- int **mbedtls_ccm_setkey** (**mbedtls_ccm_context** *ctx, **mbedtls_cipher_id_t** cipher, const unsigned char *key, unsigned int keybits)

  This function initializes the CCM context set in the ctx parameter and sets the encryption key.

- void **mbedtls_ccm_free** (**mbedtls_ccm_context** *ctx)

  This function releases and clears the specified CCM context and underlying cipher sub-context.

- int **mbedtls_ccm_encrypt_and_tag** (**mbedtls_ccm_context** *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, unsigned char *tag, size_t tag_len)

  This function encrypts a buffer using CCM.

- int **mbedtls_ccm_star_encrypt_and_tag** (**mbedtls_ccm_context** *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, unsigned char *tag, size_t tag_len)

  This function encrypts a buffer using CCM*.
- int **mbedtls_ccm_auth_decrypt** (**mbedtls_ccm_context** *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, const unsigned char *tag, size_t tag_len)

  This function performs a CCM authenticated decryption of a buffer.
- int **mbedtls_ccm_star_auth_decrypt** (**mbedtls_ccm_context** *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, const unsigned char *tag, size_t tag_len)

  This function performs a CCM*authenticated decryption of a buffer.

3.3.5 Detailed description

CCM combines Counter mode encryption with CBC-MAC authentication for 128-bit block ciphers.

Input to CCM includes the following elements:

- Payload - data that is both authenticated and encrypted.
- Associated data (Adata) - data that is authenticated but not encrypted, For example, a header.
- Nonce - A unique value that is assigned to the payload and the associated data.

  Definition of CCM: **http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf** *NISP SP 800-38C Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality* - "Counter with CBC-MAC (CCM)"

Related: *RFC-5116 An Interface and Algorithms for Authenticated Encryption*

Definition of CCM*: *IEEE 802.15.4 IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. Integer representation is fixed most-significant-octet-first order and the representation of octets is most-significant-bit-first order. This is consistent with *RFC-3610*.

3.3.6 Macro definition documentation

3.3.6.1 #define MBEDTLS_ERR_CCM_AUTH_FAILED -0x000F

Authenticated decryption failed.

3.3.6.2 #define MBEDTLS_ERR_CCM_BAD_INPUT -0x000D

Bad input parameters to the function.

3.3.6.3 #define MBEDTLS_ERR_CCM_HW_ACCEL_FAILED -0x0011

CCM hardware accelerator failed.

3.3.7 Function documentation

3.3.7.1 int mbedtls_ccm_auth_decrypt (mbedtls_ccm_context * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, const unsigned char * tag, size_t tag_len)

Parameters:

Parameter	Description
`ctx`	The CCM context to use for decryption. This must be initialized and bound to a key.
`length`	The length of the input data in bytes.
`iv`	The initialization vector (nonce). This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the nonce in bytes: 7, 8, 9, 10, 11, 12, or 13. The length L of the message length field is 15 - iv_len.
`add`	The additional data field. This must be a readable buffer of at least that add_len bytes..
`add_len`	The length of additional data in bytes. This must be less than 2^16 - 2^8.
`input`	The buffer holding the input data. If length is greater than zero, input must be a readable buffer of at least that length.
`output`	The buffer holding the output data. If length is greater than zero, output must be a writable buffer of at least that length.
`tag`	The buffer holding the authentication field. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the authentication field to generate in bytes: 4, 6, 8, 10, 12, 14 or 16.

Returns:

0 on success. This indicates that the message is authentic.

MBEDTLS_ERR_CCM_AUTH_FAILED if the tag does not match.

A cipher-specific error code on calculation failure.

3.3.7.2 int mbedtls_ccm_encrypt_and_tag (mbedtls_ccm_context * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, unsigned char * tag, size_t tag_len)

The tag is written to a separate buffer. To concatenate the tag with the output, as done in *RFC-3610 Counter with CBC-MAC (CCM)*, use tag = output + length, and make sure that the output buffer is at least length + tag_len wide.

Parameters:

Parameter	Description
`ctx`	The CCM context to use for encryption. This must be initialized and bound to a key.
`length`	The length of the input data in bytes.
`iv`	The initialization vector (nonce). This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the nonce in bytes: 7, 8, 9, 10, 11, 12, or 13. The length L of the message length field is 15 - iv_len.
`add`	The additional data field. If add_len is greater than zero, add must be a readable buffer of at least that length.
`add_len`	The length of additional data in bytes. This must be less than 2^16 - 2^8.
`input`	The buffer holding the input data. If length is greater than zero, input must be a readable buffer of at least that length.
`output`	The buffer holding the output data. If length is greater than zero, output must be a writable buffer of at least that length.
`tag`	The buffer holding the authentication field. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the authentication field to generate in bytes: 4, 6, 8, 10, 12, 14 or 16.

Returns:

0 on success.

A CCM or cipher-specific error code on failure.

3.3.7.3 void mbedtls_ccm_free (mbedtls_ccm_context * ctx)

Parameters:

Parameter	Description
`ctx`	The CCM context to clear. If this is NULL , the function has no effect. Otherwise, this must be initialized.

3.3.7.4 void mbedtls_ccm_init (mbedtls_ccm_context * ctx)

Parameters:

Parameter	Description
`ctx`	The CCM context to initialize. This must not be NULL.

3.3.7.5 int mbedtls_ccm_setkey (mbedtls_ccm_context * ctx, mbedtls_cipher_id_t cipher, const unsigned char * key, unsigned int keybits)

Parameters:

Parameter	Description
`ctx`	The CCM context to initialize. This must be an initialized context.
`cipher`	The 128-bit block cipher to use.
`key`	The encryption key. This must not be NULL.
`keybits`	The key size in bits. This must be acceptable by the cipher.

Returns:

0 on success.

A CCM or cipher-specific error code on failure.

3.3.7.6 int mbedtls_ccm_star_auth_decrypt (mbedtls_ccm_context * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, const unsigned char * tag, size_t tag_len)

When using this function in a variable tag length context, the tag length has to be decoded from iv and passed to this function as tag_len. (tag needs to be adjusted accordingly).

Parameters:

Parameter	Description
`ctx`	The CCM context to use for decryption. This must be initialized and bound to a key.
`length`	The length of the input data in bytes.
`iv`	The initialization vector (nonce). This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the nonce in bytes: 7, 8, 9, 10, 11, 12, or 13. The length L of the message length field is 15 - iv_len.

Parameter	Description
`add`	The additional data field. This must be a readable buffer of at least that add_len bytes.
`add_len`	The length of additional data in bytes. This must be less than 2^16 - 2^8.
`input`	The buffer holding the input data. If length is greater than zero, input must be a readable buffer of at least that length.
`output`	The buffer holding the output data. If length is greater than zero, output must be a writable buffer of at least that length.
`tag`	The buffer holding the authentication field. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the authentication field in bytes. 0, 4, 6, 8, 10, 12, 14 or 16.

Passing 0 as tag_len means that the message is nos longer authenticated.

Returns:

0 on success.

MBEDTLS_ERR_CCM_AUTH_FAILED if the tag does not match.

A cipher-specific error code on calculation failure.

3.3.7.7 int mbedtls_ccm_star_encrypt_and_tag (mbedtls_ccm_context * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, unsigned char * tag, size_t tag_len)

The tag is written to a separate buffer. To concatenate the tag with the output , as done in *RFC-3610 Counter with CBC-MAC (CCM)*, use tag = output + length , and make sure that the output buffer is at least length + tag_len wide.

When using this function in a variable tag length context, the tag length has to be encoded into the iv passed to this function.

Parameters:

Parameter	Description
`ctx`	The CCM context to use for encryption. This must be initialized and bound to a key.
`length`	The length of the input data in bytes.
`iv`	The initialization vector (nonce). This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the nonce in bytes: 7, 8, 9, 10, 11, 12, or 13. The length L of the message length field is 15 - iv_len.
`add`	The additional data field. This must be a readable buffer of at least add_len bytes.
`add_len`	The length of additional data in bytes. This must be less than 2^16 - 2^8.
`input`	The buffer holding the input data. If length is greater than zero, input must be a readable buffer of at least that length.
`output`	The buffer holding the output data. If length is greater than zero, output must be a writable buffer of at least that length.
`tag`	The buffer holding the authentication field. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the authentication field to generate in bytes: 0, 4, 6, 8, 10, 12, 14 or 16.

Passing 0 as tag_len means that the message is nos longer authenticated.

Returns:

0 on success.

A CCM or cipher-specific error code on failure.

3.4 chacha20.h File reference

This file contains ChaCha20 definitions and functions.

```
#include "config.h"

#include <stdint.h>

#include <stddef.h>
```

3.4.1 Data structures

- struct **mbedtls_chacha20_context**

3.4.2 Macros

- #define **MBEDTLS_ERR_CHACHA20_BAD_INPUT_DATA** -0x0051
- #define **MBEDTLS_ERR_CHACHA20_FEATURE_UNAVAILABLE** -0x0053
- #define **MBEDTLS_ERR_CHACHA20_HW_ACCEL_FAILED** -0x0055

3.4.3 typedefs

- typedef struct **mbedtls_chacha20_context** mbedtls_chacha20_context

3.4.4 Functions

- void **mbedtls_chacha20_init** (**mbedtls_chacha20_context** *ctx)

  This function initializes the specified ChaCha20 context.
- void **mbedtls_chacha20_free** (**mbedtls_chacha20_context** *ctx)

  This function releases and clears the specified ChaCha20 context.
- int **mbedtls_chacha20_setkey** (**mbedtls_chacha20_context** *ctx, const unsigned char key[32])

  This function sets the encryption/decryption key.
- int **mbedtls_chacha20_starts** (**mbedtls_chacha20_context** *ctx, const unsigned char nonce[12], uint32_t counter)

  This function sets the nonce and initial counter value.
- int **mbedtls_chacha20_update** (**mbedtls_chacha20_context** *ctx, size_t size, const unsigned char *input, unsigned char *output)

  This function encrypts or decrypts data.

- int **mbedtls_chacha20_crypt** (const unsigned char key[32], const unsigned char nonce[12], uint32_t counter, size_t size, const unsigned char *input, unsigned char *output)

  This function encrypts or decrypts data with ChaCha20 and the given key and nonce.

3.4.5 Detailed description

ChaCha20 is a stream cipher that can encrypt and decrypt information. ChaCha was created by Daniel Bernstein as a variant of its Salsa cipher **https://cr.yp.to/chacha/chacha-20080128.pdf** ChaCha20 is the variant with 20 rounds, that was also standardized in *RFC-7539*.

Author:

Daniel King **damaki.gh@gmail.com**

3.4.6 Macro definition documentation

3.4.6.1 #define MBEDTLS_ERR_CHACHA20_BAD_INPUT_DATA -0x0051

Invalid input parameter(s).

3.4.6.2 #define MBEDTLS_ERR_CHACHA20_FEATURE_UNAVAILABLE -0x0053

Feature not available. For example, s part of the API is not implemented.

3.4.6.3 #define MBEDTLS_ERR_CHACHA20_HW_ACCEL_FAILED -0x0055

Chacha20 hardware accelerator failed.

3.4.7 Function documentation

3.4.7.1 int mbedtls_chacha20_crypt (const unsigned char key[32], const unsigned char nonce[12], uint32_t counter, size_t size, const unsigned char * input, unsigned char * output)

Since ChaCha20 is a stream cipher, the same operation is used for encrypting and decrypting data.

You must never use the same (key, nonce) pair more than once. This would void any confidentiality guarantees for the messages encrypted with the same nonce and key.

The input and output pointers must either be equal or point to non-overlapping buffers.

Parameters:

Parameter	Description
`key`	The encryption/decryption key. This must be 32 bytes in length.
`nonce`	The nonce. This must be 12 bytes in size.
`counter`	The initial counter value. This is usually 0.
`size`	The length of the input data in bytes.
`input`	The buffer holding the input data. This pointer can be NULL if size == 0.
`output`	The buffer holding the output data. This must be able to hold size bytes. This pointer can be NULL if size == 0.

Returns:

0 on success.

A negative error code on failure.

3.4.7.2 void mbedtls_chacha20_free (mbedtls_chacha20_context * ctx)

Parameters:

Parameter	Description
`ctx`	The ChaCha20 context to clear. This may be NULL , in which case this function is a no-op. If it is not NULL , it must point to an initialized context.

3.4.7.3 void mbedtls_chacha20_init (mbedtls_chacha20_context * ctx)

It must be the first API called before using the context.

It is usually followed by calls to **mbedtls_chacha20_setkey()** and **mbedtls_chacha20_starts()** , then one or more calls to to **mbedtls_chacha20_update()** , and finally to **mbedtls_chacha20_free()**.

Parameters:

Parameter	Description
`ctx`	The ChaCha20 context to initialize. This must not be NULL.

3.4.7.4 int mbedtls_chacha20_setkey (mbedtls_chacha20_context * ctx, const unsigned char key[32])

After using this function, you must also call **mbedtls_chacha20_starts()** to set a nonce before you start encrypting/decrypting data with mbedtls_chacha_update().

Parameters:

Parameter	Description
`ctx`	The ChaCha20 context to which the key should be bound. It must be initialized.
`key`	The encryption/decryption key. This must be 32 bytes in length.

Returns:

0 on success.

MBEDTLS_ERR_CHACHA20_BAD_INPUT_DATA if ctx or key is NULL.

3.4.7.5 int mbedtls_chacha20_starts (mbedtls_chacha20_context * ctx, const unsigned char nonce[12], uint32_t counter)

A ChaCha20 context can be re-used with the same key by calling this function to change the nonce.

You must never use the same nonce twice with the same key. This would void any confidentiality guarantees for the messages encrypted with the same nonce and key.

Parameters:

Parameter	Description
`ctx`	The ChaCha20 context to which the nonce should be bound. It must be initialized and bound to a key.
`nonce`	The nonce. This must be 12 bytes in size.
`counter`	The initial counter value. This is usually 0.

Returns:

0 on success.

MBEDTLS_ERR_CHACHA20_BAD_INPUT_DATA if ctx or nonce is NULL.

3.4.7.6 int mbedtls_chacha20_update (mbedtls_chacha20_context * ctx, size_t size, const unsigned char * input, unsigned char * output)

Since ChaCha20 is a stream cipher, the same operation is used for encrypting and decrypting data.

The input and output pointers must either be equal or point to non-overlapping buffers.

mbedtls_chacha20_setkey() and **mbedtls_chacha20_starts()** must be called at least once to setup the context before this function can be called.

This function can be called multiple times in a row in order to encrypt of decrypt data piecewise with the same key and nonce.

Parameters:

Parameter	Description
`ctx`	The ChaCha20 context to use for encryption or decryption. It must be initialized and bound to a key and nonce.
`size`	The length of the input data in bytes.
`input`	The buffer holding the input data. This pointer can be NULL if size == 0.
`output`	The buffer holding the output data. This must be able to hold size bytes. This pointer can be NULL if size == 0.

Returns:

0 on success.

A negative error code on failure.

3.5 chachapoly.h File reference

This file contains the AEAD-ChaCha20-Poly1305 definitions and functions.

```
#include "config.h"

#include "poly1305.h"

#include "chacha20.h"
```

3.5.1 Data structures

- struct **mbedtls_chachapoly_context**

3.5.2 Macros

- #define **MBEDTLS_ERR_CHACHAPOLY_BAD_STATE** -0x0054
- #define **MBEDTLS_ERR_CHACHAPOLY_AUTH_FAILED** -0x0056

3.5.3 typedefs

- typedef struct **mbedtls_chachapoly_context** mbedtls_chachapoly_context

3.5.4 Enumerations

- enum **mbedtls_chachapoly_mode_t** { **MBEDTLS_CHACHAPOLY_ENCRYPT**, **MBEDTLS_CHACHAPOLY_DECRYPT** }

3.5.5 Functions

- void **mbedtls_chachapoly_init** (**mbedtls_chachapoly_context** *ctx)

  This function initializes the specified ChaCha20-Poly1305 context.

- void **mbedtls_chachapoly_free** (**mbedtls_chachapoly_context** *ctx)

  This function releases and clears the specified ChaCha20-Poly1305 context.

- int **mbedtls_chachapoly_setkey** (**mbedtls_chachapoly_context** *ctx, const unsigned char key[32])

  This function sets the ChaCha20-Poly1305 symmetric encryption key.

- int **mbedtls_chachapoly_starts** (**mbedtls_chachapoly_context** *ctx, const unsigned char nonce[12], **mbedtls_chachapoly_mode_t** mode)

  This function starts a ChaCha20-Poly1305 encryption or decryption operation.

- int **mbedtls_chachapoly_update_aad** (**mbedtls_chachapoly_context** *ctx, const unsigned char *aad, size_t aad_len)

This function feeds additional data to be authenticated into an ongoing ChaCha20-Poly1305 operation.

- int **mbedtls_chachapoly_update** (**mbedtls_chachapoly_context** *ctx, size_t len, const unsigned char *input, unsigned char *output)

  Thus function feeds data to be encrypted or decrypted into an on-going ChaCha20-Poly1305 operation.

- int **mbedtls_chachapoly_finish** (**mbedtls_chachapoly_context** *ctx, unsigned char mac[16])

  This function finished the ChaCha20-Poly1305 operation and generates the MAC (authentication tag).

- int **mbedtls_chachapoly_encrypt_and_tag** (**mbedtls_chachapoly_context** *ctx, size_t length, const unsigned char nonce[12], const unsigned char *aad, size_t aad_len, const unsigned char *input, unsigned char *output, unsigned char tag[16])

  This function performs a complete ChaCha20-Poly1305 authenticated encryption with the previously-set key.

- int **mbedtls_chachapoly_auth_decrypt** (**mbedtls_chachapoly_context** *ctx, size_t length, const unsigned char nonce[12], const unsigned char *aad, size_t aad_len, const unsigned char tag[16], const unsigned char *input, unsigned char *output)

  This function performs a complete ChaCha20-Poly1305 authenticated decryption with the previously-set key.

3.5.6 Detailed description

ChaCha20-Poly1305 is an algorithm for Authenticated Encryption with Associated Data (AEAD) that can be used to encrypt and authenticate data. It is based on ChaCha20 and Poly1305 by Daniel Bernstein and was standardized in *RFC-7539*.

Author:

Daniel King **damaki.gh@gmail.com**

3.5.7 Macro definition documentation

3.5.7.1 #define MBEDTLS_ERR_CHACHAPOLY_AUTH_FAILED -0x0056

Authenticated decryption failed: data was not authentic.

3.5.7.2 #define MBEDTLS_ERR_CHACHAPOLY_BAD_STATE -0x0054

The requested operation is not permitted in the current state.

3.5.8 Enumeration type documentation

3.5.8.1 enum mbedtls_chachapoly_mode_t

Enumerator:

Enum	Description
`MBEDTLS_CHACHAPOLY_ENCRYPT`	The mode value for performing encryption.
`MBEDTLS_CHACHAPOLY_DECRYPT`	The mode value for performing decryption.

3.5.9 Function documentation

3.5.9.1 int mbedtls_chachapoly_auth_decrypt (mbedtls_chachapoly_context * ctx, size_t length, const unsigned char nonce[12], const unsigned char * aad, size_t aad_len, const unsigned char tag[16], const unsigned char * input, unsigned char * output)

Before using this function, you must set the key with **mbedtls_chachapoly_setkey()**.

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context to use (holds the key).
`length`	The length (in bytes) of the data to decrypt.
`nonce`	The 96 Bit (12 bytes) nonce/IV to use.
`aad`	The buffer containing the additional authenticated data (AAD). This pointer can be NULL if aad_len == 0.
`aad_len`	The length (in bytes) of the AAD data to process.
`tag`	The buffer holding the authentication tag. This must be a readable buffer of length 16 bytes.
`input`	The buffer containing the data to decrypt. This pointer can be NULL if ilen == 0.
`output`	The buffer to where the decrypted data is written. This pointer can be NULL if ilen == 0.

Returns:

0 on success.

MBEDTLS_ERR_CHACHAPOLY_AUTH_FAILED if the data was not authentic.

Another negative error code on other kinds of failure.

3.5.9.2 int mbedtls_chachapoly_encrypt_and_tag (mbedtls_chachapoly_context * ctx, size_t length, const unsigned char nonce[12], const unsigned char * aad, size_t aad_len, const unsigned char * input, unsigned char * output, unsigned char tag[16])

Before using this function, you must set the key with **mbedtls_chachapoly_setkey()**.

You must never use the same nonce twice with the same key. This would void any confidentiality and authenticity guarantees for the messages encrypted with the same nonce and key.

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context to use (holds the key). This must be initialized.
`length`	The length (in bytes) of the data to encrypt or decrypt.
`nonce`	The 96-bit (12 bytes) nonce/IV to use.
`aad`	The buffer containing the additional authenticated data (AAD). This pointer can be NULL if aad_len == 0.
`aad_len`	The length (in bytes) of the AAD data to process.
`input`	The buffer containing the data to encrypt or decrypt. This pointer can be NULL if ilen == 0.
`output`	The buffer to where the encrypted or decrypted data is written. This pointer can be NULL if ilen == 0.
`tag`	The buffer to where the computed 128-bit (16 bytes) MAC is written. This must not be NULL.

Returns:

0 on success.

A negative error code on failure.

3.5.9.3 int mbedtls_chachapoly_finish (mbedtls_chachapoly_context * ctx, unsigned char mac[16])

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context to use. This must be initialized.
`mac`	The buffer to where the 128-bit (16 bytes) MAC is written.

Decryption with the piecewise API is discouraged, see the warning on **mbedtls_chachapoly_init()**.

Returns:

0 on success.

MBEDTLS_ERR_CHACHAPOLY_BAD_STATE if the operation has not been started or has been finished.

Another negative error code on other kinds of failure.

3.5.9.4 void mbedtls_chachapoly_free (mbedtls_chachapoly_context * ctx)

Parameters:

Parameter	Description
`ctx`	The ChachaPoly context to clear. This may be NULL , in which case this function is a no-op.

3.5.9.5 void mbedtls_chachapoly_init (mbedtls_chachapoly_context * ctx)

It must be the first API called before using the context. It must be followed by a call to **mbedtls_chachapoly_setkey()** before any operation can be done, and to **mbedtls_chachapoly_free()** once all operations with that context have been finished.

In order to encrypt or decrypt full messages at once, for each message you should make a single call to mbedtls_chachapoly_crypt_and_tag() or **mbedtls_chachapoly_auth_decrypt()**.

In order to encrypt messages piecewise, for each message you should make a call to **mbedtls_chachapoly_starts()** , then 0 or more calls to **mbedtls_chachapoly_update_aad()** , then 0 or more calls to **mbedtls_chachapoly_update()** , then one call to **mbedtls_chachapoly_finish()**.

Decryption with the piecewise API is discouraged. Always use **mbedtls_chachapoly_auth_decrypt()** when possible.

If however this is not possible because the data is too large to fit in memory, you need to:

- call **mbedtls_chachapoly_starts()** and (if needed) **mbedtls_chachapoly_update_aad()** as above,
- call **mbedtls_chachapoly_update()** multiple times and ensure its output (the plaintext) is NOT used in any other way than placing it in temporary storage at this point,
- call **mbedtls_chachapoly_finish()** to compute the authentication tag and compared it in constant time to the tag received with the ciphertext.

If the tags are not equal, you must immediately discard all previous outputs of **mbedtls_chachapoly_update()** , otherwise you can now safely use the plaintext.

Parameters:

Parameter	Description
`ctx`	The ChachaPoly context to initialize. Must not be NULL.

3.5.9.6 int mbedtls_chachapoly_setkey (mbedtls_chachapoly_context * ctx, const unsigned char key[32])

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context to which the key should be bound. This must be initialized.
`key`	The 256 Bit (32 bytes) key.

Returns:

0 on success.

A negative error code on failure.

3.5.9.7 int mbedtls_chachapoly_starts (mbedtls_chachapoly_context * ctx, const unsigned char nonce[12], mbedtls_chachapoly_mode_t mode)

You must never use the same nonce twice with the same key. This would void any confidentiality and authenticity guarantees for the messages encrypted with the same nonce and key.

If the context is being used for AAD only (no data to encrypt or decrypt) then mode can be set to any value.

Decryption with the piecewise API is discouraged, see the warning on **mbedtls_chachapoly_init()**.

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context. This must be initialized and bound to a key.
`nonce`	The nonce/IV to use for the message. This must be a redable buffer of length 12 bytes.

Parameter	Description
`mode`	The operation to perform: **MBEDTLS_CHACHAPOLY_ENCRYPT** or **MBEDTLS_CHACHAPOLY_DECRYPT** (discouraged, see warning).

Returns:

0 on success.

A negative error code on failure.

3.5.9.8 int mbedtls_chachapoly_update (mbedtls_chachapoly_context * ctx, size_t len, const unsigned char * input, unsigned char * output)

The direction (encryption or decryption) depends on the mode that was given when calling **mbedtls_chachapoly_starts()**.

You may call this function multiple times to process an arbitrary amount of data. It is permitted to call this function 0 times, if no data is to be encrypted or decrypted.

Decryption with the piecewise API is discouraged, see the warning on **mbedtls_chachapoly_init()**.

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context to use. This must be initialized.
`len`	The length (in bytes) of the data to encrypt or decrypt.
`input`	The buffer containing the data to encrypt or decrypt. This pointer can be NULL if len == 0.
`output`	The buffer to where the encrypted or decrypted data is written. This must be able to hold len bytes. This pointer can be NULL if len == 0.

Returns:

0 on success.

MBEDTLS_ERR_CHACHAPOLY_BAD_STATE if the operation has not been started or has been finished.

Another negative error code on other kinds of failure.

3.5.9.9 int mbedtls_chachapoly_update_aad (mbedtls_chachapoly_context * ctx, const unsigned char * aad, size_t aad_len)

The Additional Authenticated Data (AAD), also called Associated Data (AD) is only authenticated but not encrypted nor included in the encrypted output. It is usually transmitted separately from the ciphertext or computed locally by each party.

This function is called before data is encrypted/decrypted. I.e. call this function to process the AAD before calling **mbedtls_chachapoly_update()**.

You may call this function multiple times to process an arbitrary amount of AAD. It is permitted to call this function 0 times, if no AAD is used.

This function cannot be called any more if data has been processed by **mbedtls_chachapoly_update()** , or if the context has been finished.

Decryption with the piecewise API is discouraged, see the warning on **mbedtls_chachapoly_init()**.

Parameters:

Parameter	Description
`ctx`	The ChaCha20-Poly1305 context. This must be initialized and bound to a key.
`aad_len`	The length in bytes of the AAD. The length has no restrictions.
`aad`	Buffer containing the AAD. This pointer can be NULL if aad_len == 0.

Returns:

0 on success.

MBEDTLS_ERR_POLY1305_BAD_INPUT_DATA if ctx or aad are NULL.

MBEDTLS_ERR_CHACHAPOLY_BAD_STATE if the operations has not been started or has been finished, or if the AAD has been finished.

3.6 cipher.h File reference

This file contains an abstraction interface for use with the cipher primitives provided by the library. It provides a common interface to all of the available cipher operations.

```
#include "config.h"

#include <stddef.h>

#include "platform_util.h"
```

3.6.1 Data structures

- struct **mbedtls_cipher_info_t**
- struct **mbedtls_cipher_context_t**

3.6.2 Macros

- #define **MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE** -0x6080
- #define **MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA** -0x6100
- #define **MBEDTLS_ERR_CIPHER_ALLOC_FAILED** -0x6180
- #define **MBEDTLS_ERR_CIPHER_INVALID_PADDING** -0x6200
- #define **MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED** -0x6280
- #define **MBEDTLS_ERR_CIPHER_AUTH_FAILED** -0x6300
- #define **MBEDTLS_ERR_CIPHER_INVALID_CONTEXT** -0x6380
- #define **MBEDTLS_ERR_CIPHER_HW_ACCEL_FAILED** -0x6400
- #define **MBEDTLS_CIPHER_VARIABLE_IV_LEN** 0x01
- #define **MBEDTLS_CIPHER_VARIABLE_KEY_LEN** 0x02
- #define **MBEDTLS_MAX_IV_LENGTH** 16
- #define **MBEDTLS_MAX_BLOCK_LENGTH** 16

3.6.3 typedefs

- typedef struct **mbedtls_cipher_base_t mbedtls_cipher_base_t**
- typedef struct **mbedtls_cmac_context_t mbedtls_cmac_context_t**
- typedef struct **mbedtls_cipher_info_t mbedtls_cipher_info_t**
- typedef struct **mbedtls_cipher_context_t mbedtls_cipher_context_t**

3.6.4 Enumerations

- enum **mbedtls_cipher_id_t** { **MBEDTLS_CIPHER_ID_NONE** = 0, **MBEDTLS_CIPHER_ID_NULL**, **MBEDTLS_CIPHER_ID_AES**, **MBEDTLS_CIPHER_ID_DES**, **MBEDTLS_CIPHER_ID_3DES**, **MBEDTLS_CIPHER_ID_CAMELLIA**, **MBEDTLS_CIPHER_ID_BLOWFISH**, **MBEDTLS_CIPHER_ID_ARC4**, **MBEDTLS_CIPHER_ID_ARIA**, **MBEDTLS_CIPHER_ID_CHACHA20** }

  Supported cipher types.

- enum **mbedtls_cipher_type_t** { **MBEDTLS_CIPHER_NONE** = 0, **MBEDTLS_CIPHER_NULL**, **MBEDTLS_CIPHER_AES_128_ECB**, **MBEDTLS_CIPHER_AES_192_ECB**, **MBEDTLS_CIPHER_AES_256_ECB**, **MBEDTLS_CIPHER_AES_128_CBC**, **MBEDTLS_CIPHER_AES_192_CBC**, **MBEDTLS_CIPHER_AES_256_CBC**, **MBEDTLS_CIPHER_AES_128_CFB128**, **MBEDTLS_CIPHER_AES_192_CFB128**, **MBEDTLS_CIPHER_AES_256_CFB128**, **MBEDTLS_CIPHER_AES_128_CTR**, **MBEDTLS_CIPHER_AES_192_CTR**, **MBEDTLS_CIPHER_AES_256_CTR**, **MBEDTLS_CIPHER_AES_128_GCM**, **MBEDTLS_CIPHER_AES_192_GCM**, **MBEDTLS_CIPHER_AES_256_GCM**, **MBEDTLS_CIPHER_CAMELLIA_128_ECB**, **MBEDTLS_CIPHER_CAMELLIA_192_ECB**, **MBEDTLS_CIPHER_CAMELLIA_256_ECB**, **MBEDTLS_CIPHER_CAMELLIA_128_CBC**, **MBEDTLS_CIPHER_CAMELLIA_192_CBC**, **MBEDTLS_CIPHER_CAMELLIA_256_CBC**, **MBEDTLS_CIPHER_CAMELLIA_128_CFB128**, **MBEDTLS_CIPHER_CAMELLIA_192_CFB128**, **MBEDTLS_CIPHER_CAMELLIA_256_CFB128**, **MBEDTLS_CIPHER_CAMELLIA_128_CTR**, **MBEDTLS_CIPHER_CAMELLIA_192_CTR**, **MBEDTLS_CIPHER_CAMELLIA_256_CTR**, **MBEDTLS_CIPHER_CAMELLIA_128_GCM**, **MBEDTLS_CIPHER_CAMELLIA_192_GCM**, **MBEDTLS_CIPHER_CAMELLIA_256_GCM**, **MBEDTLS_CIPHER_DES_ECB**, **MBEDTLS_CIPHER_DES_CBC**, **MBEDTLS_CIPHER_DES_EDE_ECB**, **MBEDTLS_CIPHER_DES_EDE_CBC**, **MBEDTLS_CIPHER_DES_EDE3_ECB**, **MBEDTLS_CIPHER_DES_EDE3_CBC**, **MBEDTLS_CIPHER_BLOWFISH_ECB**, **MBEDTLS_CIPHER_BLOWFISH_CBC**, **MBEDTLS_CIPHER_BLOWFISH_CFB64**, **MBEDTLS_CIPHER_BLOWFISH_CTR**, **MBEDTLS_CIPHER_ARC4_128**, **MBEDTLS_CIPHER_AES_128_CCM**, **MBEDTLS_CIPHER_AES_192_CCM**, **MBEDTLS_CIPHER_AES_256_CCM**, **MBEDTLS_CIPHER_CAMELLIA_128_CCM**, **MBEDTLS_CIPHER_CAMELLIA_192_CCM**, **MBEDTLS_CIPHER_CAMELLIA_256_CCM**, **MBEDTLS_CIPHER_ARIA_128_ECB**, **MBEDTLS_CIPHER_ARIA_192_ECB**, **MBEDTLS_CIPHER_ARIA_256_ECB**, **MBEDTLS_CIPHER_ARIA_128_CBC**, **MBEDTLS_CIPHER_ARIA_192_CBC**, **MBEDTLS_CIPHER_ARIA_256_CBC**, **MBEDTLS_CIPHER_ARIA_128_CFB128**, **MBEDTLS_CIPHER_ARIA_192_CFB128**, **MBEDTLS_CIPHER_ARIA_256_CFB128**, **MBEDTLS_CIPHER_ARIA_128_CTR**, **MBEDTLS_CIPHER_ARIA_192_CTR**, **MBEDTLS_CIPHER_ARIA_256_CTR**, **MBEDTLS_CIPHER_ARIA_128_GCM**, **MBEDTLS_CIPHER_ARIA_192_GCM**, **MBEDTLS_CIPHER_ARIA_256_GCM**, **MBEDTLS_CIPHER_ARIA_128_CCM**, **MBEDTLS_CIPHER_ARIA_192_CCM**, **MBEDTLS_CIPHER_ARIA_256_CCM**, **MBEDTLS_CIPHER_AES_128_OFB**, **MBEDTLS_CIPHER_AES_192_OFB**, **MBEDTLS_CIPHER_AES_256_OFB**,

MBEDTLS_CIPHER_AES_128_XTS, **MBEDTLS_CIPHER_AES_256_XTS**, **MBEDTLS_CIPHER_CHACHA20**, **MBEDTLS_CIPHER_CHACHA20_POLY1305** }

Supported {cipher type, cipher mode} pairs.

- enum **mbedtls_cipher_mode_t** { **MBEDTLS_MODE_NONE** = 0, **MBEDTLS_MODE_ECB**, **MBEDTLS_MODE_CBC**, **MBEDTLS_MODE_CFB**, **MBEDTLS_MODE_OFB**, **MBEDTLS_MODE_CTR**, **MBEDTLS_MODE_GCM**, **MBEDTLS_MODE_STREAM**, **MBEDTLS_MODE_CCM**, **MBEDTLS_MODE_XTS**, **MBEDTLS_MODE_CHACHAPOLY** }
- enum **mbedtls_cipher_padding_t** { **MBEDTLS_PADDING_PKCS7** = 0, **MBEDTLS_PADDING_ONE_AND_ZEROS**, **MBEDTLS_PADDING_ZEROS_AND_LEN**, **MBEDTLS_PADDING_ZEROS**, **MBEDTLS_PADDING_NONE** }
- enum **mbedtls_operation_t** { MBEDTLS_OPERATION_NONE = -1, MBEDTLS_DECRYPT = 0, MBEDTLS_ENCRYPT }
- enum { **MBEDTLS_KEY_LENGTH_NONE** = 0, **MBEDTLS_KEY_LENGTH_DES** = 64, **MBEDTLS_KEY_LENGTH_DES_EDE** = 128, **MBEDTLS_KEY_LENGTH_DES_EDE3** = 192 }

3.6.5 Functions

- const int ***mbedtls_cipher_list** (void)

  This function retrieves the list of ciphers supported by the generic cipher module.

- const **mbedtls_cipher_info_t** ***mbedtls_cipher_info_from_string** (const char *cipher_name)

  This function retrieves the cipher-information structure associated with the given cipher name.

- const **mbedtls_cipher_info_t** ***mbedtls_cipher_info_from_type** (const **mbedtls_cipher_type_t** cipher_type)

  This function retrieves the cipher-information structure associated with the given cipher type.

- const **mbedtls_cipher_info_t** ***mbedtls_cipher_info_from_values** (const **mbedtls_cipher_id_t** cipher_id, int key_bitlen, const **mbedtls_cipher_mode_t** mode)

  This function retrieves the cipher-information structure associated with the given cipher ID, key size and mode.

- void **mbedtls_cipher_init** (**mbedtls_cipher_context_t** *ctx)

  This function initializes a cipher_context as NONE.

- void **mbedtls_cipher_free** (**mbedtls_cipher_context_t** *ctx)

  This function frees and clears the cipher-specific context of ctx. Freeing ctx itself remains the responsibility of the caller.

- int **mbedtls_cipher_setup** (**mbedtls_cipher_context_t** *ctx, const **mbedtls_cipher_info_t** *cipher_info)

This function initializes and fills the cipher-context structure with the appropriate values. It also clears the structure.

- int **mbedtls_cipher_setkey** (**mbedtls_cipher_context_t** *ctx, const unsigned char *key, int key_bitlen, const **mbedtls_operation_t** operation)

  This function sets the key to use with the given context.

- int **mbedtls_cipher_set_iv** (**mbedtls_cipher_context_t** *ctx, const unsigned char *iv, size_t iv_len)

  This function sets the initialization vector (IV) or nonce.

- int **mbedtls_cipher_reset** (**mbedtls_cipher_context_t** *ctx)

  This function resets the cipher state.

- int **mbedtls_cipher_update** (**mbedtls_cipher_context_t** *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)

  The generic cipher update function. It encrypts or decrypts using the given cipher context. Writes as many block-sized blocks of data as possible to output. Any data that cannot be written immediately is either added to the next block, or flushed when **mbedtls_cipher_finish()** is called. Exception: For MBEDTLS_MODE_ECB, expects a single block in size. For example, 16 bytes for AES.

- int **mbedtls_cipher_finish** (**mbedtls_cipher_context_t** *ctx, unsigned char *output, size_t *olen)

  The generic cipher finalization function. If data still needs to be flushed from an incomplete block, the data contained in it is padded to the size of the last block, and written to the output buffer.

- int **mbedtls_cipher_crypt** (**mbedtls_cipher_context_t** *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)

  The generic all-in-one encryption/decryption function, for all ciphers except AEAD constructs.

3.6.6 Detailed description

Author:

Adriaan de Jong **dejong@fox-it.com**

3.6.7 Macro definition documentation

3.6.7.1 #define MBEDTLS_CIPHER_VARIABLE_IV_LEN 0x01

Cipher accepts IVs of variable length.

3.6.7.2 #define MBEDTLS_CIPHER_VARIABLE_KEY_LEN 0x02

Cipher accepts keys of variable length.

3.6.7.3 #define MBEDTLS_ERR_CIPHER_ALLOC_FAILED -0x6180

Failed to allocate memory.

3.6.7.4 #define MBEDTLS_ERR_CIPHER_AUTH_FAILED -0x6300

Authentication failed (for AEAD modes).

3.6.7.5 #define MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA -0x6100

Bad input parameters.

3.6.7.6 #define MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE -0x6080

The selected feature is not available.

3.6.7.7 #define MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED -0x6280

Decryption of block requires a full block.

3.6.7.8 #define MBEDTLS_ERR_CIPHER_HW_ACCEL_FAILED -0x6400

Cipher hardware accelerator failed.

3.6.7.9 #define MBEDTLS_ERR_CIPHER_INVALID_CONTEXT -0x6380

The context is invalid. For example, because it was freed.

3.6.7.10 #define MBEDTLS_ERR_CIPHER_INVALID_PADDING -0x6200

Input data contains invalid padding and is rejected.

3.6.7.11 #define MBEDTLS_MAX_BLOCK_LENGTH 16

Maximum block size of any cipher, in bytes.

3.6.7.12 #define MBEDTLS_MAX_IV_LENGTH 16

Maximum length of any IV, in bytes.

3.6.8 typedef documentation

3.6.8.1 typedef struct mbedtls_cipher_base_t mbedtls_cipher_base_t

Base cipher information (opaque struct).

3.6.8.2 typedef struct mbedtls_cipher_context_t mbedtls_cipher_context_t

Generic cipher context.

3.6.8.3 typedef struct mbedtls_cipher_info_t mbedtls_cipher_info_t

Cipher information. Allows calling cipher functions in a generic way.

3.6.8.4 typedef struct mbedtls_cmac_context_t mbedtls_cmac_context_t

CMAC context (opaque struct).

3.6.9 Enumeration type documentation

3.6.9.1 anonymous enum

Enumerator:

Enum	Description
`MBEDTLS_KEY_LENGTH_NONE`	Undefined key length.
`MBEDTLS_KEY_LENGTH_DES`	Key length, in bits (including parity), for DES keys.
`MBEDTLS_KEY_LENGTH_DES_EDE`	Key length in bits, including parity, for DES in two-key EDE.
`MBEDTLS_KEY_LENGTH_DES_EDE3`	Key length in bits, including parity, for DES in three-key EDE.

3.6.9.2 enum mbedtls_cipher_id_t

RC4 and DES are considered weak ciphers and their use constitutes a security risk. Arm recommends considering stronger ciphers instead.

Enumerator:

Enum	Description
`MBEDTLS_CIPHER_ID_NONE`	Placeholder to mark the end of cipher ID lists.
`MBEDTLS_CIPHER_ID_NULL`	The identity cipher, treated as a stream cipher.
`MBEDTLS_CIPHER_ID_AES`	The AES cipher.

Enum	Description
`MBEDTLS_CIPHER_ID_DES`	The DES cipher.
`MBEDTLS_CIPHER_ID_3DES`	The Triple DES cipher.
`MBEDTLS_CIPHER_ID_CAMELLIA`	The Camellia cipher.
`MBEDTLS_CIPHER_ID_BLOWFISH`	The Blowfish cipher.
`MBEDTLS_CIPHER_ID_ARC4`	The RC4 cipher.
`MBEDTLS_CIPHER_ID_ARIA`	The Aria cipher.
`MBEDTLS_CIPHER_ID_CHACHA20`	The ChaCha20 cipher.

3.6.9.3 enum mbedtls_cipher_mode_t

Supported cipher modes.

Enumerator:

Enum	Description
`MBEDTLS_MODE_NONE`	None.
`MBEDTLS_MODE_ECB`	The ECB cipher mode.
`MBEDTLS_MODE_CBC`	The CBC cipher mode.
`MBEDTLS_MODE_CFB`	The CFB cipher mode.
`MBEDTLS_MODE_OFB`	The OFB cipher mode.
`MBEDTLS_MODE_CTR`	The CTR cipher mode.
`MBEDTLS_MODE_GCM`	The GCM cipher mode.
`MBEDTLS_MODE_STREAM`	The stream cipher mode.
`MBEDTLS_MODE_CCM`	The CCM cipher mode.
`MBEDTLS_MODE_XTS`	The XTS cipher mode.
`MBEDTLS_MODE_CHACHAPOLY`	The ChaCha-Poly cipher mode.

3.6.9.4 enum mbedtls_cipher_padding_t

Supported cipher padding types.

Enumerator:

Enum	Description
`MBEDTLS_PADDING_PKCS7`	*PKCS #7* padding (default).
`MBEDTLS_PADDING_ONE_AND_ZEROS`	*ISO/IEC 7816-4* padding.
`MBEDTLS_PADDING_ZEROS_AND_LEN`	*ANSI X9.23* padding.
`MBEDTLS_PADDING_ZEROS`	Zero padding (not reversible).

Enum	Description
`MBEDTLS_PADDING_NONE`	Never pad (full blocks only).

3.6.9.5 **enum** mbedtls_cipher_type_t

RC4 and DES are considered weak ciphers and their use constitutes a security risk. Arm recommends considering stronger ciphers instead.

Enumerator:

Enum	Description
`MBEDTLS_CIPHER_NONE`	Placeholder to mark the end of cipher-pair lists.
`MBEDTLS_CIPHER_NULL`	The identity stream cipher.
`MBEDTLS_CIPHER_AES_128_ECB`	AES cipher with 128-bit ECB mode.
`MBEDTLS_CIPHER_AES_192_ECB`	AES cipher with 192-bit ECB mode.
`MBEDTLS_CIPHER_AES_256_ECB`	AES cipher with 256-bit ECB mode.
`MBEDTLS_CIPHER_AES_128_CBC`	AES cipher with 128-bit CBC mode.
`MBEDTLS_CIPHER_AES_192_CBC`	AES cipher with 192-bit CBC mode.
`MBEDTLS_CIPHER_AES_256_CBC`	AES cipher with 256-bit CBC mode.
`MBEDTLS_CIPHER_AES_128_CFB128`	AES cipher with 128-bit CFB128 mode.
`MBEDTLS_CIPHER_AES_192_CFB128`	AES cipher with 192-bit CFB128 mode.
`MBEDTLS_CIPHER_AES_256_CFB128`	AES cipher with 256-bit CFB128 mode.
`MBEDTLS_CIPHER_AES_128_CTR`	AES cipher with 128-bit CTR mode.
`MBEDTLS_CIPHER_AES_192_CTR`	AES cipher with 192-bit CTR mode.
`MBEDTLS_CIPHER_AES_256_CTR`	AES cipher with 256-bit CTR mode.
`MBEDTLS_CIPHER_AES_128_GCM`	AES cipher with 128-bit GCM mode.
`MBEDTLS_CIPHER_AES_192_GCM`	AES cipher with 192-bit GCM mode.
`MBEDTLS_CIPHER_AES_256_GCM`	AES cipher with 256-bit GCM mode.
`MBEDTLS_CIPHER_CAMELLIA_128_ECB`	Camellia cipher with 128-bit ECB mode.
`MBEDTLS_CIPHER_CAMELLIA_192_ECB`	Camellia cipher with 192-bit ECB mode.
`MBEDTLS_CIPHER_CAMELLIA_256_ECB`	Camellia cipher with 256-bit ECB mode.
`MBEDTLS_CIPHER_CAMELLIA_128_CBC`	Camellia cipher with 128-bit CBC mode.
`MBEDTLS_CIPHER_CAMELLIA_192_CBC`	Camellia cipher with 192-bit CBC mode.
`MBEDTLS_CIPHER_CAMELLIA_256_CBC`	Camellia cipher with 256-bit CBC mode.
`MBEDTLS_CIPHER_CAMELLIA_128_CFB128`	Camellia cipher with 128-bit CFB128 mode.
`MBEDTLS_CIPHER_CAMELLIA_192_CFB128`	Camellia cipher with 192-bit CFB128 mode.

Enum	Description
`MBEDTLS_CIPHER_CAMELLIA_256_CFB128`	Camellia cipher with 256-bit CFB128 mode.
`MBEDTLS_CIPHER_CAMELLIA_128_CTR`	Camellia cipher with 128-bit CTR mode.
`MBEDTLS_CIPHER_CAMELLIA_192_CTR`	Camellia cipher with 192-bit CTR mode.
`MBEDTLS_CIPHER_CAMELLIA_256_CTR`	Camellia cipher with 256-bit CTR mode.
`MBEDTLS_CIPHER_CAMELLIA_128_GCM`	Camellia cipher with 128-bit GCM mode.
`MBEDTLS_CIPHER_CAMELLIA_192_GCM`	Camellia cipher with 192-bit GCM mode.
`MBEDTLS_CIPHER_CAMELLIA_256_GCM`	Camellia cipher with 256-bit GCM mode.
`MBEDTLS_CIPHER_DES_ECB`	DES cipher with ECB mode.
`MBEDTLS_CIPHER_DES_CBC`	DES cipher with CBC mode.
`MBEDTLS_CIPHER_DES_EDE_ECB`	DES cipher with EDE ECB mode.
`MBEDTLS_CIPHER_DES_EDE_CBC`	DES cipher with EDE CBC mode.
`MBEDTLS_CIPHER_DES_EDE3_ECB`	DES cipher with EDE3 ECB mode.
`MBEDTLS_CIPHER_DES_EDE3_CBC`	DES cipher with EDE3 CBC mode.
`MBEDTLS_CIPHER_BLOWFISH_ECB`	Blowfish cipher with ECB mode.
`MBEDTLS_CIPHER_BLOWFISH_CBC`	Blowfish cipher with CBC mode.
`MBEDTLS_CIPHER_BLOWFISH_CFB64`	Blowfish cipher with CFB64 mode.
`MBEDTLS_CIPHER_BLOWFISH_CTR`	Blowfish cipher with CTR mode.
`MBEDTLS_CIPHER_ARC4_128`	RC4 cipher with 128-bit mode.
`MBEDTLS_CIPHER_AES_128_CCM`	AES cipher with 128-bit CCM mode.
`MBEDTLS_CIPHER_AES_192_CCM`	AES cipher with 192-bit CCM mode.
`MBEDTLS_CIPHER_AES_256_CCM`	AES cipher with 256-bit CCM mode.
`MBEDTLS_CIPHER_CAMELLIA_128_CCM`	Camellia cipher with 128-bit CCM mode.
`MBEDTLS_CIPHER_CAMELLIA_192_CCM`	Camellia cipher with 192-bit CCM mode.
`MBEDTLS_CIPHER_CAMELLIA_256_CCM`	Camellia cipher with 256-bit CCM mode.
`MBEDTLS_CIPHER_ARIA_128_ECB`	Aria cipher with 128-bit key and ECB mode.
`MBEDTLS_CIPHER_ARIA_192_ECB`	Aria cipher with 192-bit key and ECB mode.
`MBEDTLS_CIPHER_ARIA_256_ECB`	Aria cipher with 256-bit key and ECB mode.
`MBEDTLS_CIPHER_ARIA_128_CBC`	Aria cipher with 128-bit key and CBC mode.
`MBEDTLS_CIPHER_ARIA_192_CBC`	Aria cipher with 192-bit key and CBC mode.
`MBEDTLS_CIPHER_ARIA_256_CBC`	Aria cipher with 256-bit key and CBC mode.
`MBEDTLS_CIPHER_ARIA_128_CFB128`	Aria cipher with 128-bit key and CFB-128 mode.
`MBEDTLS_CIPHER_ARIA_192_CFB128`	Aria cipher with 192-bit key and CFB-128 mode.

Enum	Description
`MBEDTLS_CIPHER_ARIA_256_CFB128`	Aria cipher with 256-bit key and CFB-128 mode.
`MBEDTLS_CIPHER_ARIA_128_CTR`	Aria cipher with 128-bit key and CTR mode.
`MBEDTLS_CIPHER_ARIA_192_CTR`	Aria cipher with 192-bit key and CTR mode.
`MBEDTLS_CIPHER_ARIA_256_CTR`	Aria cipher with 256-bit key and CTR mode.
`MBEDTLS_CIPHER_ARIA_128_GCM`	Aria cipher with 128-bit key and GCM mode.
`MBEDTLS_CIPHER_ARIA_192_GCM`	Aria cipher with 192-bit key and GCM mode.
`MBEDTLS_CIPHER_ARIA_256_GCM`	Aria cipher with 256-bit key and GCM mode.
`MBEDTLS_CIPHER_ARIA_128_CCM`	Aria cipher with 128-bit key and CCM mode.
`MBEDTLS_CIPHER_ARIA_192_CCM`	Aria cipher with 192-bit key and CCM mode.
`MBEDTLS_CIPHER_ARIA_256_CCM`	Aria cipher with 256-bit key and CCM mode.
`MBEDTLS_CIPHER_AES_128_OFB`	AES 128-bit cipher in OFB mode.
`MBEDTLS_CIPHER_AES_192_OFB`	AES 192-bit cipher in OFB mode.
`MBEDTLS_CIPHER_AES_256_OFB`	AES 256-bit cipher in OFB mode.
`MBEDTLS_CIPHER_AES_128_XTS`	AES 128-bit cipher in XTS block mode.
`MBEDTLS_CIPHER_AES_256_XTS`	AES 256-bit cipher in XTS block mode.
`MBEDTLS_CIPHER_CHACHA20`	ChaCha20 stream cipher.
`MBEDTLS_CIPHER_CHACHA20_POLY1305`	ChaCha20-Poly1305 AEAD cipher.

3.6.9.6 enum mbedtls_operation_t

Type of operation.

3.6.10 Function documentation

3.6.10.1 int mbedtls_cipher_crypt (mbedtls_cipher_context_t * ctx, const unsigned char * iv, size_t iv_len, const unsigned char * input, size_t ilen, unsigned char * output, size_t * olen)

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized.

Parameter	Description
`iv`	The IV to use, or NONCE_COUNTER for CTR-mode ciphers. This must be a readable buffer of at least iv_len bytes.
`iv_len`	The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV.
`input`	The buffer holding the input data. This must be a readable buffer of at least ilen bytes.
`ilen`	The length of the input data in bytes.
`output`	The buffer for the output data. This must be able to hold at least ilen + block_size. This must not be the same buffer as input.
`olen`	The length of the output data, to be updated with the actual number of bytes written. This must not be NULL.

Some ciphers do not use IVs nor nonce. For these ciphers, use iv = NULL and iv_len = 0.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED on decryption expecting a full block but not receiving one.

MBEDTLS_ERR_CIPHER_INVALID_PADDING on invalid padding while decrypting.

A cipher-specific error code on failure.

3.6.10.2 int mbedtls_cipher_finish (mbedtls_cipher_context_t * ctx, unsigned char * output, size_t * olen)

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized and bound to a key.
`output`	The buffer to write data to. This needs to be a writable buffer of at least block_size bytes.
`olen`	The length of the data written to the output buffer. This may not be NULL.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED on decryption expecting a full block but not receiving one.

MBEDTLS_ERR_CIPHER_INVALID_PADDING on invalid padding while decrypting.

A cipher-specific error code on failure.

3.6.10.3 void mbedtls_cipher_free (mbedtls_cipher_context_t * ctx)

Parameters:

Parameter	Description
`ctx`	The context to be freed. If this is NULL , the function has no effect, otherwise this must point to an initialized context.

3.6.10.4 const mbedtls_cipher_info_t*mbedtls_cipher_info_from_string (const char * cipher_name)

Parameters:

Parameter	Description
`cipher_name`	Name of the cipher to search for. This must not be NULL.

Returns:

The cipher information structure associated with the given cipher_name.

NULL if the associated cipher information is not found.

3.6.10.5 const mbedtls_cipher_info_t*mbedtls_cipher_info_from_type (const mbedtls_cipher_type_t cipher_type)

Parameters:

Parameter	Description
`cipher_type`	Type of the cipher to search for.

Returns:

The cipher information structure associated with the given cipher_type.

NULL if the associated cipher information is not found.

3.6.10.6 const mbedtls_cipher_info_t*mbedtls_cipher_info_from_values (const mbedtls_cipher_id_t cipher_id, int key_bitlen, const mbedtls_cipher_mode_t mode)

Parameters:

Parameter	Description
`cipher_id`	The ID of the cipher to search for. For example, **MBEDTLS_CIPHER_ID_AES**.
`key_bitlen`	The length of the key in bits.
`mode`	The cipher mode. For example, **MBEDTLS_MODE_CBC**.

Returns:

The cipher information structure associated with the given cipher_id.

NULL if the associated cipher information is not found.

3.6.10.7 void mbedtls_cipher_init (mbedtls_cipher_context_t * ctx)

Parameters:

Parameter	Description
`ctx`	The context to be initialized. This must not be NULL.

3.6.10.8 const int*mbedtls_cipher_list (void)

Returns:

A statically-allocated array of ciphers. The last entry is zero.

3.6.10.9 int mbedtls_cipher_reset (mbedtls_cipher_context_t * ctx)

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

3.6.10.10 int mbedtls_cipher_set_iv (mbedtls_cipher_context_t * ctx, const unsigned char * iv, size_t iv_len)

Some ciphers do not use IVs nor nonce. For these ciphers, this function has no effect.

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized and bound to a cipher information structure.
`iv`	The IV to use, or NONCE_COUNTER for CTR-mode ciphers. This must be a readable buffer of at least iv_len bytes.
`iv_len`	The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

3.6.10.11 int mbedtls_cipher_setkey (mbedtls_cipher_context_t * ctx, const unsigned char * key, int key_bitlen, const mbedtls_operation_t operation)

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized and bound to a cipher information structure.
`key`	The key to use. This must be a readable buffer of at least key_bitlen Bits.
`key_bitlen`	The key length to use, in Bits.
`operation`	The operation that the key will be used for: #MBEDTLS_ENCRYPT or #MBEDTLS_DECRYPT.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

A cipher-specific error code on failure.

3.6.10.12 int mbedtls_cipher_setup (mbedtls_cipher_context_t * ctx, const mbedtls_cipher_info_t * cipher_info)

Parameters:

Parameter	Description
`ctx`	The context to initialize. This must be initialized.
`cipher_info`	The cipher to use.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_CIPHER_ALLOC_FAILED if allocation of the cipher-specific context fails.

3.6.10.13 int mbedtls_cipher_update (mbedtls_cipher_context_t * ctx, const unsigned char * input, size_t ilen, unsigned char * output, size_t * olen)

If the underlying cipher is used in GCM mode, all calls to this function, except for the last one before **mbedtls_cipher_finish()**, must have ilen as a multiple of the block size of the cipher.

Parameters:

Parameter	Description
`ctx`	The generic cipher context. This must be initialized and bound to a key.
`input`	The buffer holding the input data. This must be a readable buffer of at least ilen bytes.
`ilen`	The length of the input data.
`output`	The buffer for the output data. This must be able to hold at least ilen + block_size. This must not be the same buffer as input.
`olen`	The length of the output data, to be updated with the actual number of bytes written. This must not be NULL.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE on an unsupported mode for a cipher.

A cipher-specific error code on failure.

3.7 cmac.h File reference

This file contains CMAC definitions and functions.

```
#include "config.h"
```

```
#include "cipher.h"
```

3.7.1 Data structures

- struct **mbedtls_cmac_context_t**

3.7.2 Macros

- #define **MBEDTLS_ERR_CMAC_HW_ACCEL_FAILED** -0x007A
- #define MBEDTLS_AES_BLOCK_SIZE 16
- #define MBEDTLS_DES3_BLOCK_SIZE 8
- #define **MBEDTLS_CIPHER_BLKSIZE_MAX** 8

3.7.3 Functions

- int **mbedtls_cipher_cmac_starts** (**mbedtls_cipher_context_t** *ctx, const unsigned char *key, size_t keybits)

  This function sets the CMAC key, and prepares to authenticate the input data. Must be called with an initialized cipher context.

- int **mbedtls_cipher_cmac_update** (**mbedtls_cipher_context_t** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing CMAC computation.

- int **mbedtls_cipher_cmac_finish** (**mbedtls_cipher_context_t** *ctx, unsigned char *output)

  This function finishes the CMAC operation, and writes the result to the output buffer.

- int **mbedtls_cipher_cmac_reset** (**mbedtls_cipher_context_t** *ctx)

  This function prepares the authentication of another message with the same key as the previous CMAC operation.

- int **mbedtls_cipher_cmac** (const **mbedtls_cipher_info_t** *cipher_info, const unsigned char *key, size_t keylen, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the full generic CMAC on the input buffer with the provided key.

3.7.4 Detailed description

The Cipher-based Message Authentication Code (CMAC) Mode for Authentication is defined in *RFC-4493 The AES-CMAC Algorithm*.

3.7.5 Macro definition documentation

3.7.5.1 #define MBEDTLS_CIPHER_BLKSIZE_MAX 8

The longest block used by CMAC is that of 3DES.

3.7.5.2 #define MBEDTLS_ERR_CMAC_HW_ACCEL_FAILED -0x007A

CMAC hardware accelerator failed.

3.7.6 Function documentation

3.7.6.1 int mbedtls_cipher_cmac (const mbedtls_cipher_info_t * cipher_info, const unsigned char * key, size_t keylen, const unsigned char * input, size_t ilen, unsigned char * output)

The function allocates the context, performs the calculation, and frees the context.

The CMAC result is calculated as output = generic CMAC(cmac key, input buffer).

Parameters:

Parameter	Description
`cipher_info`	The cipher information.
`key`	The CMAC key.
`keylen`	The length of the CMAC key in bits.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The buffer for the generic CMAC result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA if parameter verification fails.

3.7.6.2 int mbedtls_cipher_cmac_finish (mbedtls_cipher_context_t * ctx, unsigned char * output)

It is called after **mbedtls_cipher_cmac_update()**. It can be followed by **mbedtls_cipher_cmac_reset()** and **mbedtls_cipher_cmac_update()**, or **mbedtls_cipher_free()**.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.
`output`	The output buffer for the CMAC checksum result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA if parameter verification fails.

3.7.6.3 int mbedtls_cipher_cmac_reset (mbedtls_cipher_context_t * ctx)

It is called after **mbedtls_cipher_cmac_finish()** and before **mbedtls_cipher_cmac_update()**.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA if parameter verification fails.

3.7.6.4 int mbedtls_cipher_cmac_starts (mbedtls_cipher_context_t * ctx, const unsigned char * key, size_t keybits)

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation, initialized as one of the following types: MBEDTLS_CIPHER_AES_128_ECB, MBEDTLS_CIPHER_AES_192_ECB, MBEDTLS_CIPHER_AES_256_ECB, or MBEDTLS_CIPHER_DES_EDE3_ECB.
`key`	The CMAC key.
`keybits`	The length of the CMAC key in bits. Must be supported by the cipher.

Returns:

0 on success.

A cipher-specific error code on failure.

3.7.6.5 int mbedtls_cipher_cmac_update (mbedtls_cipher_context_t * ctx, const unsigned char * input, size_t ilen)

It is called between **mbedtls_cipher_cmac_starts()** or **mbedtls_cipher_cmac_reset()**, and **mbedtls_cipher_cmac_finish()**. Can be called repeatedly.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA if parameter verification fails.

3.8 ctr_drbg.h File reference

This file contains CTR_DRBG definitions and functions.

```
#include "config.h"

#include "aes.h"
```

3.8.1 Data structures

- struct **mbedtls_ctr_drbg_context**

  The CTR_DRBG context structure.

3.8.2 Macros

- #define **MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED** -0x0034
- #define **MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG** -0x0036
- #define **MBEDTLS_ERR_CTR_DRBG_INPUT_TOO_BIG** -0x0038
- #define **MBEDTLS_ERR_CTR_DRBG_FILE_IO_ERROR** -0x003A
- #define **MBEDTLS_CTR_DRBG_BLOCKSIZE** 16
- #define **MBEDTLS_CTR_DRBG_KEYSIZE** 32
- #define **MBEDTLS_CTR_DRBG_KEYBITS** (**MBEDTLS_CTR_DRBG_KEYSIZE** *8)
- #define **MBEDTLS_CTR_DRBG_SEEDLEN** (**MBEDTLS_CTR_DRBG_KEYSIZE** + **MBEDTLS_CTR_DRBG_BLOCKSIZE**)
- #define **MBEDTLS_CTR_DRBG_PR_OFF** 0
- #define **MBEDTLS_CTR_DRBG_PR_ON** 1
- #define MBEDTLS_DEPRECATED

3.8.3 Module settings

The configuration options you can set for this module are in this section. Either change them in config.h or define them using the compiler command line.

- o #define **MBEDTLS_CTR_DRBG_ENTROPY_LEN** 32
- o #define **MBEDTLS_CTR_DRBG_RESEED_INTERVAL** 10000
- o #define **MBEDTLS_CTR_DRBG_MAX_INPUT** 256
- o #define **MBEDTLS_CTR_DRBG_MAX_REQUEST** 1024
- o #define **MBEDTLS_CTR_DRBG_MAX_SEED_INPUT** 384

3.8.3.1 Typedefs

- typedef struct **mbedtls_ctr_drbg_context** **mbedtls_ctr_drbg_context**

  The CTR_DRBG context structure.

3.8.3.2 Functions

- void **mbedtls_ctr_drbg_init** (**mbedtls_ctr_drbg_context** *ctx)

  This function initializes the CTR_DRBG context, and prepares it for **mbedtls_ctr_drbg_seed()** or **mbedtls_ctr_drbg_free()**.

- int **mbedtls_ctr_drbg_seed** (**mbedtls_ctr_drbg_context** *ctx, int(*f_entropy)(void *, unsigned char *, size_t), void *p_entropy, const unsigned char *custom, size_t len)

  This function seeds and sets up the CTR_DRBG entropy source for future reseeds.

- void **mbedtls_ctr_drbg_free** (**mbedtls_ctr_drbg_context** *ctx)

  This function clears CTR_CRBG context data.

- void **mbedtls_ctr_drbg_set_prediction_resistance** (**mbedtls_ctr_drbg_context** *ctx, int resistance)

  This function turns prediction resistance on or off. The default value is off.

- void **mbedtls_ctr_drbg_set_entropy_len** (**mbedtls_ctr_drbg_context** *ctx, size_t len)

  This function sets the amount of entropy grabbed on each seed or reseed. The default value is **MBEDTLS_CTR_DRBG_ENTROPY_LEN**.

- void **mbedtls_ctr_drbg_set_reseed_interval** (**mbedtls_ctr_drbg_context** *ctx, int interval)

  This function sets the reseed interval. The default value is **MBEDTLS_CTR_DRBG_RESEED_INTERVAL**.

- int **mbedtls_ctr_drbg_reseed** (**mbedtls_ctr_drbg_context** *ctx, const unsigned char *additional, size_t len)

  This function reseeds the CTR_DRBG context, that is extracts data from the entropy source.

- int **mbedtls_ctr_drbg_update_ret** (**mbedtls_ctr_drbg_context** *ctx, const unsigned char *additional, size_t add_len)

  This function updates the state of the CTR_DRBG context.

- int **mbedtls_ctr_drbg_random_with_add** (void *p_rng, unsigned char *output, size_t output_len, const unsigned char *additional, size_t add_len)

  This function updates a CTR_DRBG instance with additional data and uses it to generate random data.

- int **mbedtls_ctr_drbg_random** (void *p_rng, unsigned char *output, size_t output_len)

This function uses CTR_DRBG to generate random data.

- MBEDTLS_DEPRECATED void **mbedtls_ctr_drbg_update** (**mbedtls_ctr_drbg_context** *ctx, const unsigned char *additional, size_t add_len)

  This function updates the state of the CTR_DRBG context.

- int mbedtls_ctr_drbg_seed_entropy_len (**mbedtls_ctr_drbg_context** *, int(*)(void *, unsigned char *, size_t), void *, const unsigned char *, size_t, size_t)

3.8.3.3 Detailed Description

CTR_DRBG is a standardized way of building a PRNG from a block-cipher in counter mode operation, as defined in *NIST SP 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators*.

The Mbed TLS implementation of CTR_DRBG uses AES-256 (default) or AES-128 as the underlying block cipher.

Using 128-bit keys for CTR_DRBG limits the security of generated keys and operations that use random values generated to 128-bit security.

3.8.3.4 Macro Definition Documentation

3.8.3.4.1 #define MBEDTLS_CTR_DRBG_BLOCKSIZE 16

The block size used by the cipher.

3.8.3.4.2 #define MBEDTLS_CTR_DRBG_ENTROPY_LEN 32

Amount of entropy used per seed by default:

- 48 with SHA-512.
- 32 with SHA-256.

3.8.3.4.3 #define MBEDTLS_CTR_DRBG_KEYBITS (MBEDTLS_CTR_DRBG_KEYSIZE *8)

The key size for the DRBG operation, in bits.

3.8.3.4.4 #define MBEDTLS_CTR_DRBG_KEYSIZE 32

The key size used by the cipher (compile-time choice: 256 bits).

3.8.3.4.5 #define MBEDTLS_CTR_DRBG_MAX_INPUT 256

The maximum number of additional input bytes.

3.8.3.4.6 #define MBEDTLS_CTR_DRBG_MAX_REQUEST 1024

The maximum number of requested bytes per call.

3.8.3.4.7 #define MBEDTLS_CTR_DRBG_MAX_SEED_INPUT 384

The maximum size of seed or reseed buffer.

3.8.3.4.8 #define MBEDTLS_CTR_DRBG_PR_OFF 0

Prediction resistance is disabled.

3.8.3.4.9 #define MBEDTLS_CTR_DRBG_PR_ON 1

Prediction resistance is enabled.

3.8.3.4.10 #define MBEDTLS_CTR_DRBG_RESEED_INTERVAL 10000

The interval before reseed is performed by default.

3.8.3.4.11 #define MBEDTLS_CTR_DRBG_SEEDLEN (MBEDTLS_CTR_DRBG_KEYSIZE + MBEDTLS_CTR_DRBG_BLOCKSIZE)

The seed length, calculated as (counter + AES key).

3.8.3.4.12 #define MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED -0x0034

The entropy source failed.

3.8.3.4.13 #define MBEDTLS_ERR_CTR_DRBG_FILE_IO_ERROR -0x003A

Read or write error in file.

3.8.3.4.14 #define MBEDTLS_ERR_CTR_DRBG_INPUT_TOO_BIG -0x0038

The input (entropy + additional data) is too large.

3.8.3.4.15 #define MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG -0x0036

The requested random buffer length is too big.

3.8.3.5 Function Documentation

3.8.3.5.1 void mbedtls_ctr_drbg_free (mbedtls_ctr_drbg_context * ctx)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context to clear.

3.8.3.5.2 void mbedtls_ctr_drbg_init (mbedtls_ctr_drbg_context * ctx)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context to initialize.

3.8.3.5.3 int mbedtls_ctr_drbg_random (void * p_rng, unsigned char * output, size_t output_len)

The function automatically reseeds if the reseed counter is exceeded.

Parameters:

Parameter	Description
`p_rng`	The CTR_DRBG context. This must be a pointer to a **mbedtls_ctr_drbg_context** structure.
`output`	The buffer to fill.
`output_len`	The length of the buffer.

Returns:

0 on success.

MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED or **MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG** on failure.

3.8.3.5.4 int mbedtls_ctr_drbg_random_with_add (void * p_rng, unsigned char * output, size_t output_len, const unsigned char * additional, size_t add_len)

The function automatically reseeds if the reseed counter is exceeded.

Parameters:

Parameter	Description
`p_rng`	The CTR_DRBG context. This must be a pointer to a **mbedtls_ctr_drbg_context** structure.
`output`	The buffer to fill.
`output_len`	The length of the buffer.
`additional`	Additional data to update. Can be NULL.
`add_len`	The length of the additional data.

Returns:

0 on success.

MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED or **MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG** on failure.

3.8.3.5.5 int mbedtls_ctr_drbg_reseed (mbedtls_ctr_drbg_context * ctx, const unsigned char * additional, size_t len)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`additional`	Additional data to add to the state. Can be NULL.
`len`	The length of the additional data.

Returns:

0 on success.

MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED on failure.

3.8.3.5.6 int mbedtls_ctr_drbg_seed (mbedtls_ctr_drbg_context * ctx, int(*)(void *, unsigned char *, size_t) f_entropy, void * p_entropy, const unsigned char * custom, size_t len)

Personalization data can be provided in addition to the more generic entropy source, to make this instantiation as unique as possible.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context to seed.
`f_entropy`	The entropy callback, taking as arguments the p_entropy context, the buffer to fill, and the length of the buffer.

Parameter	Description
`p_entropy`	The entropy context.
`custom`	Personalization data, that is device-specific identifiers. Can be NULL.
`len`	The length of the personalization data.

Returns:

0 on success.

MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED on failure.

3.8.3.5.7 void mbedtls_ctr_drbg_set_entropy_len (mbedtls_ctr_drbg_context * ctx, size_t len)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`len`	The amount of entropy to grab.

3.8.3.5.8 void mbedtls_ctr_drbg_set_prediction_resistance (mbedtls_ctr_drbg_context * ctx, int resistance)

If enabled, entropy is gathered at the beginning of every call to **mbedtls_ctr_drbg_random_with_add()**. Only use this if your entropy source has sufficient throughput.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`resistance`	**MBEDTLS_CTR_DRBG_PR_ON** or **MBEDTLS_CTR_DRBG_PR_OFF**.

3.8.3.5.9 void mbedtls_ctr_drbg_set_reseed_interval (mbedtls_ctr_drbg_context * ctx, int interval)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`interval`	The reseed interval.

3.8.3.5.10 MBEDTLS_DEPRECATED void mbedtls_ctr_drbg_update (mbedtls_ctr_drbg_context * ctx, const unsigned char * additional, size_t add_len)

Deprecated:

Superseded by **mbedtls_ctr_drbg_update_ret()** in 2.16.0.

If add_len is greater than **MBEDTLS_CTR_DRBG_MAX_SEED_INPUT**, only the first **MBEDTLS_CTR_DRBG_MAX_SEED_INPUT** bytes are used. The remaining bytes are silently discarded.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`additional`	The data to update the state with.
`add_len`	Length of additional data.

3.8.3.5.11 int mbedtls_ctr_drbg_update_ret (mbedtls_ctr_drbg_context * ctx, const unsigned char * additional, size_t add_len)

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`additional`	The data to update the state with.
`add_len`	Length of additional in bytes. This must be at most **MBEDTLS_CTR_DRBG_MAX_SEED_INPUT**.

Returns:

0 on success.

MBEDTLS_ERR_CTR_DRBG_INPUT_TOO_BIG if add_len is more than **MBEDTLS_CTR_DRBG_MAX_SEED_INPUT**.

An error from the underlying AES cipher on failure.

3.9 dhm.h File reference

This file contains Diffie-Hellman-Merkle (DHM) key exchange definitions and functions.

```
#include "config.h"

#include "bignum.h"
```

3.9.1 Data structures

- struct **mbedtls_dhm_context**

  The DHM context structure.

3.9.2 Macros

- #define **MBEDTLS_ERR_DHM_BAD_INPUT_DATA** -0x3080
- #define **MBEDTLS_ERR_DHM_READ_PARAMS_FAILED** -0x3100
- #define **MBEDTLS_ERR_DHM_MAKE_PARAMS_FAILED** -0x3180
- #define **MBEDTLS_ERR_DHM_READ_PUBLIC_FAILED** -0x3200
- #define **MBEDTLS_ERR_DHM_MAKE_PUBLIC_FAILED** -0x3280
- #define **MBEDTLS_ERR_DHM_CALC_SECRET_FAILED** -0x3300
- #define **MBEDTLS_ERR_DHM_INVALID_FORMAT** -0x3380
- #define **MBEDTLS_ERR_DHM_ALLOC_FAILED** -0x3400
- #define **MBEDTLS_ERR_DHM_FILE_IO_ERROR** -0x3480
- #define **MBEDTLS_ERR_DHM_HW_ACCEL_FAILED** -0x3500
- #define **MBEDTLS_ERR_DHM_SET_GROUP_FAILED** -0x3580
- #define **MBEDTLS_DHM_RFC5114_MODP_2048_P**
- #define **MBEDTLS_DHM_RFC5114_MODP_2048_G**
- #define **MBEDTLS_DHM_RFC3526_MODP_2048_P**
- #define **MBEDTLS_DHM_RFC3526_MODP_2048_G** MBEDTLS_DEPRECATED_STRING_CONSTANT("02")
- #define **MBEDTLS_DHM_RFC3526_MODP_3072_P**
- #define **MBEDTLS_DHM_RFC3526_MODP_3072_G** MBEDTLS_DEPRECATED_STRING_CONSTANT("02")
- #define **MBEDTLS_DHM_RFC3526_MODP_4096_P**

- #define **MBEDTLS_DHM_RFC3526_MODP_4096_G** MBEDTLS_DEPRECATED_STRING_CONSTANT("02")
- #define MBEDTLS_DHM_RFC3526_MODP_2048_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_2048_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC3526_MODP_3072_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_3072_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC3526_MODP_4096_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_4096_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE2048_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE2048_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE3072_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE3072_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE4096_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE4096_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE6144_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE6144_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE8192_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE8192_G_BIN { 0x02 }

3.9.3 typedefs

- typedef struct **mbedtls_dhm_context mbedtls_dhm_context**

  The DHM context structure.

3.9.4 Functions

- void **mbedtls_dhm_init** (**mbedtls_dhm_context** *ctx)

  This function initializes the DHM context.

- int **mbedtls_dhm_read_params** (**mbedtls_dhm_context** *ctx, unsigned char **p, const unsigned char *end)

  This function parses the DHM parameters in a TLS ServerKeyExchange handshake message (DHM modulus, generator, and public key).

- int **mbedtls_dhm_make_params** (**mbedtls_dhm_context** *ctx, int x_size, unsigned char *output, size_t *olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a DHM key pair and exports its public part together with the DHM parameters in the format used in a TLS ServerKeyExchange handshake message.

- int **mbedtls_dhm_set_group** (**mbedtls_dhm_context** *ctx, const mbedtls_mpi *P, const mbedtls_mpi *G)

  This function sets the prime modulus and generator.
- int **mbedtls_dhm_read_public** (**mbedtls_dhm_context** *ctx, const unsigned char *input, size_t ilen)

  This function imports the raw public value of the peer.
- int **mbedtls_dhm_make_public** (**mbedtls_dhm_context** *ctx, int x_size, unsigned char *output, size_t olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function creates a DHM key pair and exports the raw public key in big-endian format.
- int **mbedtls_dhm_calc_secret** (**mbedtls_dhm_context** *ctx, unsigned char *output, size_t output_size, size_t *olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function derives and exports the shared secret (G^Y)^X mod P.
- void **mbedtls_dhm_free** (**mbedtls_dhm_context** *ctx)

  This function frees and clears the components of a DHM context.

3.9.5 Detailed description

Diffie-Hellman-Merkle (DHM) key exchange is defined in *RFC-2631 Diffie-Hellman Key Agreement Method* and *PKCS #3 Public-Key Cryptography Standards Diffie Hellman Key Agreement Standard*.

RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) defines a number of standardized Diffie-Hellman groups for IKE.

RFC-5114 Additional Diffie-Hellman Groups for Use with IETF Standards defines a number of standardized Diffie-Hellman groups that can be used.

The security of the DHM key exchange relies on the proper choice of prime modulus - optimally, it should be a safe prime. The usage of non-safe primes both decreases the difficulty of the underlying discrete logarithm problem and can lead to small subgroup attacks leaking private exponent bits when invalid public keys are used and not detected. This is especially relevant if the same DHM parameters are reused for multiple key exchanges as in static DHM, while the criticality of small-subgroup attacks is lower for ephemeral DHM.

For performance reasons, the code does neither perform primality nor safe primality tests, nor the expensive checks for invalid subgroups. Moreover, even if these were performed, non-standardized primes cannot be trusted because of the possibility of backdoors that can't be effectively checked for.

Diffie-Hellman-Merkle is therefore a security risk when not using standardized primes generated using a trustworthy ("nothing up my sleeve") method, such as the *RFC-3526/RFC-7919* primes. In the TLS protocol, DH parameters need to be negotiated, so using the default primes systematically is not always an option. If possible, use Elliptic

Curve Diffie-Hellman (ECDH), which has better performance, and for which the TLS protocol mandates the use of standard parameters.

3.9.6 Macro definition documentation

3.9.6.1 #define MBEDTLS_DHM_RFC3526_MODP_2048_G MBEDTLS_DEPRECATED_STRING_CONSTANT("02")

The hexadecimal presentation of the chosen generator of the 2048-bit MODP Group, as defined in *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

3.9.6.2 #define MBEDTLS_DHM_RFC3526_MODP_2048_P

```
MBEDTLS_DEPRECATED_STRING_CONSTANT(                            \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"      \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"      \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"      \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"      \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"      \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"      \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"      \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"      \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"      \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"      \
        "15728E5A8AACAA68FFFFFFFFFFFFFFFF")
```

The hexadecimal presentation of the prime underlying the 2048-bit MODP Group, as defined in *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

Deprecated:

The hex-encoded primes from *RFC-3526* are deprecated and superseded by the corresponding macros providing them as binary constants. Their hex-encoded constants are likely to be removed in a future version of the library.

3.9.6.3 #define MBEDTLS_DHM_RFC3526_MODP_3072_G MBEDTLS_DEPRECATED_STRING_CONSTANT("02")

The hexadecimal presentation of the chosen generator of the 3072-bit MODP Group, as defined in RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).

3.9.6.4 #define MBEDTLS_DHM_RFC3526_MODP_3072_P

```
MBEDTLS_DEPRECATED_STRING_CONSTANT(                          \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"      \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"      \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"      \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"      \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"      \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"      \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"      \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"      \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"      \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"      \
        "15728E5A8AAAC42DAD33170D04507A33A85521ABDF1CBA64"      \
        "ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7"      \
        "ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6B"      \
        "F12FFA06D98A0864D87602733EC86A64521F2B18177B200C"      \
        "BBE117577A615D6C770988C0BAD946E208E24FA074E5AB31"      \
        "43DB5BFCE0FD108E4B82D120A93AD2CAFFFFFFFFFFFFFFFF")
```

The hexadecimal presentation of the prime underlying the 3072-bit MODP Group, as defined in *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

3.9.6.5 #define MBEDTLS_DHM_RFC3526_MODP_4096_G MBEDTLS_DEPRECATED_STRING_CONSTANT("02")

The hexadecimal presentation of the chosen generator of the 4096-bit MODP Group, as defined in *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

3.9.6.6 #define MBEDTLS_DHM_RFC3526_MODP_4096_P

```
MBEDTLS_DEPRECATED_STRING_CONSTANT(                       \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"   \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"   \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"   \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"   \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"   \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"   \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"   \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"   \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"   \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"   \
        "15728E5A8AAAC42DAD33170D04507A33A85521ABDF1CBA64"   \
        "ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7"   \
        "ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6B"   \
        "F12FFA06D98A0864D87602733EC86A64521F2B18177B200C"   \
        "BBE117577A615D6C770988C0BAD946E208E24FA074E5AB31"   \
        "43DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D7"   \
        "88719A10BDBA5B2699C327186AF4E23C1A946834B6150BDA"   \
        "2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC141FBECAA6"   \
        "287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED"   \
        "1F612970CEE2D7AFB81BDD762170481CD0069127D5B05AA9"   \
        "93B4EA988D8FDDC186FFB7DC90A6C08F4DF435C934063199"   \
        "FFFFFFFFFFFFFFFF")
```

The hexadecimal presentation of the prime underlying the 4096-bit MODP Group, as defined in *RFC-3526 More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*.

3.9.6.7 #define MBEDTLS_DHM_RFC5114_MODP_2048_G

```
MBEDTLS_DEPRECATED_STRING_CONSTANT(                          \
        "AC4032EF4F2D9AE39DF30B5C8FFDAC506CDEBE7B89998CAF"      \
        "74866A08CFE4FFE3A6824A4E10B9A6F0DD921F01A70C4AFA"      \
        "AB739D7700C29F52C57DB17C620A8652BE5E9001A8D66AD7"      \
        "C17669101999024AF4D027275AC1348BB8A762D0521BC98A"      \
        "E247150422EA1ED409939D54DA7460CDB5F6C6B250717CBE"      \
        "F180EB34118E98D119529A45D6F834566E3025E316A330EF"      \
        "BB77A86F0C1AB15B051AE3D428C8F8ACB70A8137150B8EEB"      \
        "10E183EDD19963DDD9E263E4770589EF6AA21E7F5F2FF381"      \
        "B539CCE3409D13CD566AFBB48D6C019181E1BCFE94B30269"      \
        "EDFE72FE9B6AA4BD7B5A0F1C71CFFF4C19C418E1F6EC0179"      \
        "81BC087F2A7065B384B890D3191F2BFA")
```

The hexadecimal presentation of the chosen generator of the 2048-bit MODP Group with 224-bit Prime Order Subgroup, as defined in *RFC-5114 Additional Diffie-Hellman Groups for Use with IETF Standards*.

3.9.6.8 #define MBEDTLS_DHM_RFC5114_MODP_2048_P

```
MBEDTLS_DEPRECATED_STRING_CONSTANT(                          \
        "AD107E1E9123A9D0D660FAA79559C51FA20D64E5683B9FD1"      \
        "B54B1597B61D0A75E6FA141DF95A56DBAF9A3C407BA1DF15"      \
        "EB3D688A309C180E1DE6B85A1274A0A66D3F8152AD6AC212"      \
        "9037C9EDEFDA4DF8D91E8FEF55B7394B7AD5B7D0B6C12207"      \
        "C9F98D11ED34DBF6C6BA0B2C8BBC27BE6A00E0A0B9C49708"      \
        "B3BF8A317091883681286130BC8985DB1602E714415D9330"      \
        "278273C7DE31EFDC7310F7121FD5A07415987D9ADC0A486D"      \
        "CDF93ACC44328387315D75E198C641A480CD86A1B9E587E8"      \
        "BE60E69CC928B2B9C52172E413042E9B23F10B0E16E79763"      \
        "C9B53DCF4BA80A29E3FB73C16B8E75B97EF363E2FFA31F71"      \
        "CF9DE5384E71B81C0AC4DFFE0C10E64F")
```

RFC-3526, *RFC-5114* and *RFC-7919* standardize a number of Diffie-Hellman groups, some of which are included here for use within the SSL/TLS module and the user's convenience when configuring the Diffie-Hellman parameters by hand through mbedtls_ssl_conf_dh_param.

The following lists the source of the above groups in the standards:

- *RFC-5114* section 2.2: 2048-bit MODP Group with 224-bit Prime Order Subgroup
- *RFC-3526* section 3: 2048-bit MODP Group
- *RFC-3526* section 4: 3072-bit MODP Group
- *RFC-3526* section 5: 4096-bit MODP Group
- *RFC-7919* section A.1: ffdhe2048
- *RFC-7919* section A.2: ffdhe3072
- *RFC-7919* section A.3: ffdhe4096
- *RFC-7919* section A.4: ffdhe6144
- *RFC-7919* section A.5: ffdhe8192

The constants with suffix "_p" denote the chosen prime moduli, while the constants with suffix "_g" denote the chosen generator of the associated prime field.

The constants further suffixed with "_bin" are provided in binary format, while all other constants represent null-terminated strings holding the hexadecimal presentation of the respective numbers.

The primes from *RFC-3526* and *RFC-7919* have been generating by the following trust-worthy procedure:

- Fix N in { 2048, 3072, 4096, 6144, 8192 } and consider the N-bit number the first and last 64 bits are all 1, and the remaining N - 128 bits of which are 0x7ff...ff.
- Add the smallest multiple of the first N - 129 bits of the binary expansion of pi (for *RFC-5236*) or e (for *RFC-7919*) to this intermediate bit-string such that the resulting integer is a safe-prime.
- The result is the respective *RFC-3526*/*RFC-7919* prime, and the corresponding generator is always chosen to be 2 (which is a square for these prime, hence the corresponding subgroup has order (p-1)/2 and avoids leaking a bit in the private exponent).

The origin of the primes in RFC-5114 is not documented and their use therefore constitutes a security risk!

Deprecated**:**

- The hex-encoded primes from RFC-5114 are deprecated and are likely to be removed in a future version of the library without replacement.
- The hexadecimal presentation of the prime underlying the 2048-bit MODP Group with 224-bit Prime Order Subgroup, as defined in *RFC-5114 Additional Diffie-Hellman Groups for Use with IETF Standards*.

3.9.6.9 #define MBEDTLS_ERR_DHM_ALLOC_FAILED -0x3400

Allocation of memory failed.

3.9.6.10 #define MBEDTLS_ERR_DHM_BAD_INPUT_DATA -0x3080

Bad input parameters.

3.9.6.11 #define MBEDTLS_ERR_DHM_CALC_SECRET_FAILED -0x3300

Calculation of the DHM secret failed.

3.9.6.12 #define MBEDTLS_ERR_DHM_FILE_IO_ERROR -0x3480

Read or write of file failed.

3.9.6.13 #define MBEDTLS_ERR_DHM_HW_ACCEL_FAILED -0x3500

DHM hardware accelerator failed.

3.9.6.14 #define MBEDTLS_ERR_DHM_INVALID_FORMAT -0x3380

The ASN.1 data is not formatted correctly.

3.9.6.15 #define MBEDTLS_ERR_DHM_MAKE_PARAMS_FAILED -0x3180

Making of the DHM parameters failed.

3.9.6.16 #define MBEDTLS_ERR_DHM_MAKE_PUBLIC_FAILED -0x3280

Making of the public value failed.

3.9.6.17 #define MBEDTLS_ERR_DHM_READ_PARAMS_FAILED -0x3100

Reading of the DHM parameters failed.

3.9.6.18 #define MBEDTLS_ERR_DHM_READ_PUBLIC_FAILED -0x3200

Reading of the public values failed.

3.9.6.19 #define MBEDTLS_ERR_DHM_SET_GROUP_FAILED -0x3580

Setting the modulus and generator failed.

3.9.7 Function documentation

3.9.7.1 int mbedtls_dhm_calc_secret (mbedtls_dhm_context * ctx, unsigned char * output, size_t output_size, size_t * olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

If f_rng is not NULL , it is used to blind the input as a countermeasure against timing attacks. Blinding is used only if our private key X is re-used, and not used otherwise. We recommend always passing a non-NULL f_rng argument.

Parameters:

Parameter	Description
`ctx`	The DHM context to use. This must be initialized and have its own private key generated and the peer's public key imported.
`output`	The buffer to write the generated shared key to. This must be a writable buffer of size output_size bytes.
`output_size`	The size of the destination buffer. This must be at least the size of ctx->len (the size of P).
`olen`	On exit, holds the actual number of bytes written.
`f_rng`	The RNG function, for blinding purposes. This may b NULL if blinding isn't needed.
`p_rng`	The RNG context. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.9.7.2 void mbedtls_dhm_free (mbedtls_dhm_context * ctx)

Parameters:

Parameter	Description
`ctx`	The DHM context to free and clear. This may be NULL , in which case this function is a no-op. If it is not NULL , it must point to an initialized DHM context.

3.9.7.3 void mbedtls_dhm_init (mbedtls_dhm_context * ctx)

Parameters:

Parameter	Description
`ctx`	The DHM context to initialize.

3.9.7.4 int mbedtls_dhm_make_params (mbedtls_dhm_context * ctx, int x_size, unsigned char * output, size_t * olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function assumes that the DHM parameters ctx->P and ctx->G have already been properly set. For that, use **mbedtls_dhm_set_group()** below in conjunction with mbedtls_mpi_read_binary() and mbedtls_mpi_read_string().

In a TLS handshake, this is the how the server generates and exports its DHM key material.

Parameters:

Parameter	Description
`ctx`	The DHM context to use. This must be initialized and have the DHM parameters set. It may or may not already have imported the peer's public key.
`x_size`	The private key size in bytes.
`olen`	The address at which to store the number of bytes written on success. This must not be NULL.
`output`	The destination buffer. This must be a writable buffer of sufficient size to hold the reduced binary presentation of the modulus, the generator and the public key, each wrapped with a 2-byte length field. It is the responsibility of the caller to ensure that enough space is available. Refer to mbedtls_mpi_size() to computing the byte-size of an MPI.
`f_rng`	The RNG function. Must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context parameter.

Returns:

0 on success.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.9.7.5 int mbedtls_dhm_make_public (mbedtls_dhm_context * ctx, int x_size, unsigned char * output, size_t olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

The destination buffer is always fully written so as to contain a big-endian representation of G^X mod P. If it is larger than ctx->len , it is padded accordingly with zero-bytes at the beginning.

Parameters:

Parameter	Description
`ctx`	The DHM context to use. This must be initialized and have the DHM parameters set. It may or may not already have imported the peer's public key.
`x_size`	The private key size in bytes.
`output`	The destination buffer. This must be a writable buffer of size olen bytes.
`olen`	The length of the destination buffer. This must be at least equal to ctx->len (the size of P).
`f_rng`	The RNG function. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.9.7.6 int mbedtls_dhm_read_params (mbedtls_dhm_context * ctx, unsigned char ** p, const unsigned char * end)

In a TLS handshake, this is the how the client sets up its DHM context from the server's public DHM key material.

Parameters:

Parameter	Description
`ctx`	The DHM context to use. This must be initialized.
`p`	On input, *p must be the start of the input buffer. On output, *p is updated to point to the end of the data that has been read. On success, this is the first byte past the end of the ServerKeyExchange parameters. On error, this is the point at which an error has been detected, which is usually not useful except to debug failures.
`end`	The end of the input buffer.

Returns:

0 on success.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.9.7.7 int mbedtls_dhm_read_public (mbedtls_dhm_context * ctx, const unsigned char * input, size_t ilen)

In a TLS handshake, this is how the server imports the Client's public DHM key.

Parameters:

Parameter	Description
`ctx`	The DHM context to use. This must be initialized and have its DHM parameters set, e.g. via **mbedtls_dhm_set_group()**. It may or may not already have generated its own private key.
`input`	The input buffer containing the G^Y value of the peer. This must be a readable buffer of size ilen bytes.
`ilen`	The size of the input buffer input in bytes.

Returns:

0 on success.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.9.7.8 int mbedtls_dhm_set_group (mbedtls_dhm_context * ctx, const mbedtls_mpi * P, const mbedtls_mpi * G)

This function can be used to set ctx->P , ctx->G in preparation for **mbedtls_dhm_make_params()**.

Parameters:

Parameter	Description
`ctx`	The DHM context to configure. This must be initialized.
`P`	The MPI holding the DHM prime modulus. This must be an initialized MPI.
`G`	The MPI holding the DHM generator. This must be an initialized MPI.

Returns:

0 if successful.

An MBEDTLS_ERR_DHM_XXX error code on failure.

3.10 ecdh.h File reference

This file contains ECDH definitions and functions.

```
#include "config.h"
```

```
#include "ecp.h"
```

3.10.1 Data structures

- struct **mbedtls_ecdh_context**

  The ECDH context structure.

3.10.2 typedefs

- typedef struct **mbedtls_ecdh_context mbedtls_ecdh_context**

  The ECDH context structure.

3.10.3 Enumerations

- enum **mbedtls_ecdh_side** { **MBEDTLS_ECDH_OURS**, **MBEDTLS_ECDH_THEIRS** }

3.10.4 Functions

- int **mbedtls_ecdh_gen_public** (**mbedtls_ecp_group** *grp, mbedtls_mpi *d, **mbedtls_ecp_point** *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECDH keypair on an elliptic curve.

- int **mbedtls_ecdh_compute_shared** (**mbedtls_ecp_group** *grp, mbedtls_mpi *z, const **mbedtls_ecp_point** *Q, const mbedtls_mpi *d, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the shared secret.

- void **mbedtls_ecdh_init** (**mbedtls_ecdh_context** *ctx)

  This function initializes an ECDH context.

- int **mbedtls_ecdh_setup** (**mbedtls_ecdh_context** *ctx, **mbedtls_ecp_group_id** grp_id)

  This function sets up the ECDH context with the information given.

- void **mbedtls_ecdh_free** (**mbedtls_ecdh_context** *ctx)

  This function frees a context.

- int **mbedtls_ecdh_make_params** (**mbedtls_ecdh_context** *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

This function generates an EC key pair and exports its in the format used in a TLS ServerKeyExchange handshake message.

- int **mbedtls_ecdh_read_params** (**mbedtls_ecdh_context** *ctx, const unsigned char **buf, const unsigned char *end)

  This function parses the ECDHE parameters in a TLS ServerKeyExchange handshake message.

- int **mbedtls_ecdh_get_params** (**mbedtls_ecdh_context** *ctx, const **mbedtls_ecp_keypair** *key, **mbedtls_ecdh_side** side)

  This function sets up an ECDH context from an EC key.

- int **mbedtls_ecdh_make_public** (**mbedtls_ecdh_context** *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a public key and exports it as a TLS ClientKeyExchange payload.

- int **mbedtls_ecdh_read_public** (**mbedtls_ecdh_context** *ctx, const unsigned char *buf, size_t blen)

  This function parses and processes the ECDHE payload of a TLS ClientKeyExchange message.

- int **mbedtls_ecdh_calc_secret** (**mbedtls_ecdh_context** *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function derives and exports the shared secret.

3.10.5 Detailed description

The Elliptic Curve Diffie-Hellman (ECDH) protocol is an anonymous key agreement protocol allowing two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

For more information, see *NIST SP 800-56A Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography Rev. 2*.

3.10.6 typedef documentation

3.10.6.1 typedef struct mbedtls_ecdh_context mbedtls_ecdh_context

Performing multiple operations concurrently on the same ECDSA context is not supported; objects of this type should not be shared between multiple threads.

3.10.7 Enumeration type documentation

3.10.7.1 enum mbedtls_ecdh_side

Defines the source of the imported EC key.

Enumerator:

Enum	Description
`MBEDTLS_ECDH_OURS`	Our key.
`MBEDTLS_ECDH_THEIRS`	The key of the peer.

3.10.8 Function documentation

3.10.8.1 int mbedtls_ecdh_calc_secret (mbedtls_ecdh_context * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This is the last function used by both TLS client and servers.

If f_rng is not NULL, it is used to implement countermeasures against side-channel attacks. For more information, see **mbedtls_ecp_mul()**.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDH context to use. This must be initialized and have its own private key generated and the peer's public key imported.
`olen`	The address at which to store the total number of bytes written on success. This must not be NULL.
`buf`	The buffer to write the generated shared key to. This must be a writable buffer of size blen bytes.
`blen`	The length of the destination buffer buf in bytes.
`f_rng`	The RNG function, for blinding purposes. This may b NULL if blinding isn't needed.
`p_rng`	The RNG context. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.2 int mbedtls_ecdh_compute_shared (mbedtls_ecp_group * grp, mbedtls_mpi * z, const mbedtls_ecp_point * Q, const mbedtls_mpi * d, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function performs the second of two core computations implemented during the ECDH key exchange. The first core computation is performed by **mbedtls_ecdh_gen_public()**.

See also:

ecp.h

If f_rng is not NULL, it is used to implement countermeasures against side-channel attacks. For more information, see **mbedtls_ecp_mul()**.

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have domain parameters loaded, for example through mbedtls_ecp_load() or **mbedtls_ecp_tls_read_group()**.
`z`	The destination MPI (shared secret). This must be initialized.
`Q`	The public key from another party. This must be initialized.
`d`	Our secret exponent (private key). This must be initialized.
`f_rng`	The RNG function. This may be NULL if randomization of intermediate results during the ECP computations is not needed (discouraged). See the documentation of **mbedtls_ecp_mul()** for more.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context argument.

Returns:

0 on success.

Another MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.10.8.3 void mbedtls_ecdh_free (mbedtls_ecdh_context * ctx)

Parameters:

Parameter	Description
`ctx`	The context to free. This may be NULL , in which case this function does nothing. If it is not NULL , it must point to an initialized ECDH context.

3.10.8.4 int mbedtls_ecdh_gen_public (mbedtls_ecp_group * grp, mbedtls_mpi * d, mbedtls_ecp_point * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function performs the first of two core computations implemented during the ECDH key exchange. The second core computation is performed by **mbedtls_ecdh_compute_shared()**.

See also:

ecp.h

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have domain parameters loaded, for example through mbedtls_ecp_load() or **mbedtls_ecp_tls_read_group()**.
`d`	The destination MPI (private key). This must be initialized.
`Q`	The destination point (public key). This must be initialized.
`f_rng`	The RNG function to use. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL in case f_rng doesn't need a context argument.

Returns:

0 on success.

Another MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.10.8.5 int mbedtls_ecdh_get_params (mbedtls_ecdh_context * ctx, const mbedtls_ecp_keypair * key, mbedtls_ecdh_side side)

It is used by clients and servers in place of the ServerKeyEchange for static ECDH, and imports ECDH parameters from the EC key information of a certificate.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDH context to set up. This must be initialized.
`key`	The EC key to use. This must be initialized.
`side`	Defines the source of the key. Possible values are: **MBEDTLS_ECDH_OURS**: The key is ours. **MBEDTLS_ECDH_THEIRS**: The key is that of the peer.

Returns:

0 on success.

Another MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.6 void mbedtls_ecdh_init (mbedtls_ecdh_context * ctx)

Parameters:

Parameter	Description
`ctx`	The ECDH context to initialize. This must not be NULL.

3.10.8.7 int mbedtls_ecdh_make_params (mbedtls_ecdh_context * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This is the second function used by a TLS server for ECDHE ciphersuites. (It is called after **mbedtls_ecdh_setup()**.)

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDH context to use. This must be initialized and bound to a group, for example via **mbedtls_ecdh_setup()**.
`olen`	The address at which to store the number of bytes written.
`buf`	The destination buffer. This must be a writable buffer of length blen bytes.
`blen`	The length of the destination buffer buf in bytes.
`f_rng`	The RNG function to use. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL in case f_rng doesn't need a context argument.

Returns:

0 on success.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.8 int mbedtls_ecdh_make_public (mbedtls_ecdh_context * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This is the second function used by a TLS client for ECDH(E) ciphersuites.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDH context to use. This must be initialized and bound to a group, the latter usually by **mbedtls_ecdh_read_params()**.
`olen`	The address at which to store the number of bytes written. This must not be NULL.
`buf`	The destination buffer. This must be a writable buffer of length blen bytes.
`blen`	The size of the destination buffer buf in bytes.
`f_rng`	The RNG function to use. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL in case f_rng doesn't need a context argument.

Returns:

0 on success.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.9 int mbedtls_ecdh_read_params (mbedtls_ecdh_context * ctx, const unsigned char ** buf, const unsigned char * end)

In a TLS handshake, this is the how the client sets up its ECDHE context from the server's public ECDHE key material.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDHE context to use. This must be initialized.
`buf`	On input, *buf must be the start of the input buffer. On output, *buf is updated to point to the end of the data that has been read. On success, this is the first byte past the end of the ServerKeyExchange parameters. On error, this is the point at which an error has been detected, which is usually not useful except to debug failures.
`end`	The end of the input buffer.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.10 int mbedtls_ecdh_read_public (mbedtls_ecdh_context * ctx, const unsigned char * buf, size_t blen)

This is the third function used by a TLS server for ECDH(E) ciphersuites. (It is called after **mbedtls_ecdh_setup()** and **mbedtls_ecdh_make_params()**.)

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDH context to use. This must be initialized and bound to a group, for example via **mbedtls_ecdh_setup()**.
`buf`	The pointer to the ClientKeyExchange payload. This must be a readable buffer of length blen bytes.
`blen`	The length of the input buffer buf in bytes.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX error code on failure.

3.10.8.11 int mbedtls_ecdh_setup (mbedtls_ecdh_context * ctx, mbedtls_ecp_group_id grp_id)

This function should be called after **mbedtls_ecdh_init()** but before **mbedtls_ecdh_make_params()**. There is no need to call this function before **mbedtls_ecdh_read_params()**.

This is the first function used by a TLS server for ECDHE ciphersuites.

Parameters:

Parameter	Description
`ctx`	The ECDH context to set up. This must be initialized.
`grp_id`	The group id of the group to set up the context for.

Returns:

0 on success.

3.11 ecdsa.h File reference

This file contains ECDSA definitions and functions.

```
#include "config.h"

#include "ecp.h"

#include "md.h"
```

3.11.1 Macros

- #define **MBEDTLS_ECDSA_MAX_LEN** (3 + 2 *(3 + MBEDTLS_ECP_MAX_BYTES))

3.11.2 typedefs

- typedef **mbedtls_ecp_keypair mbedtls_ecdsa_context**

  The ECDSA context structure.

- typedef void mbedtls_ecdsa_restart_ctx

3.11.3 Functions

- int **mbedtls_ecdsa_sign** (**mbedtls_ecp_group** *grp, mbedtls_mpi *r, mbedtls_mpi *s, const mbedtls_mpi *d, const unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the ECDSA signature of a previously-hashed message.

- int **mbedtls_ecdsa_verify** (**mbedtls_ecp_group** *grp, const unsigned char *buf, size_t blen, const **mbedtls_ecp_point** *Q, const mbedtls_mpi *r, const mbedtls_mpi *s)

  This function verifies the ECDSA signature of a previously-hashed message.

- int **mbedtls_ecdsa_write_signature** (**mbedtls_ecdsa_context** *ctx, **mbedtls_md_type_t** md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t *slen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the ECDSA signature and writes it to a buffer, serialized as defined in *RFC-4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*.

- int **mbedtls_ecdsa_write_signature_restartable** (**mbedtls_ecdsa_context** *ctx, **mbedtls_md_type_t** md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t *slen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, mbedtls_ecdsa_restart_ctx *rs_ctx)

  This function computes the ECDSA signature and writes it to a buffer, in a restartable way.

- int **mbedtls_ecdsa_read_signature** (**mbedtls_ecdsa_context** *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen)

  This function reads and verifies an ECDSA signature.
- int **mbedtls_ecdsa_read_signature_restartable** (**mbedtls_ecdsa_context** *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen, mbedtls_ecdsa_restart_ctx *rs_ctx)

  This function reads and verifies an ECDSA signature, in a restartable way.
- int **mbedtls_ecdsa_genkey** (**mbedtls_ecdsa_context** *ctx, **mbedtls_ecp_group_id** gid, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECDSA keypair on the given curve.
- int **mbedtls_ecdsa_from_keypair** (**mbedtls_ecdsa_context** *ctx, const **mbedtls_ecp_keypair** *key)

  This function sets up an ECDSA context from an EC key pair.
- void **mbedtls_ecdsa_init** (**mbedtls_ecdsa_context** *ctx)

  This function initializes an ECDSA context.
- void **mbedtls_ecdsa_free** (**mbedtls_ecdsa_context** *ctx)

  This function frees an ECDSA context.

3.11.4 Detailed description

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in *SECG SEC1 Elliptic Curve Cryptography*. The use of ECDSA for TLS is defined in *RFC-4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*.

3.11.5 Macro definition documentation

3.11.5.1 #define MBEDTLS_ECDSA_MAX_LEN (3 + 2 *(3 + MBEDTLS_ECP_MAX_BYTES))

The maximal size of an ECDSA signature in bytes.

3.11.6 typedef documentation

3.11.6.1 typedef mbedtls_ecp_keypair mbedtls_ecdsa_context

Performing multiple operations concurrently on the same ECDSA context is not supported; objects of this type should not be shared between multiple threads.

3.11.7 Function documentation

3.11.7.1 void mbedtls_ecdsa_free (mbedtls_ecdsa_context * ctx)

Parameters:

Parameter	Description
`ctx`	The ECDSA context to free. This may be NULL , in which case this function does nothing. If it is not NULL , it must be initialized.

3.11.7.2 int mbedtls_ecdsa_from_keypair (mbedtls_ecdsa_context * ctx, const mbedtls_ecp_keypair * key)

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDSA context to setup. This must be initialized.
`key`	The EC key to use. This must be initialized and hold a private-public key pair or a public key. In the former case, the ECDSA context may be used for signature creation and verification after this call. In the latter case, it may be used for signature verification.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX code on failure.

3.11.7.3 int mbedtls_ecdsa_genkey (mbedtls_ecdsa_context * ctx, mbedtls_ecp_group_id gid, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDSA context to store the keypair in. This must be initialized.
`gid`	The elliptic curve to use. One of the various MBEDTLS_ECP_DP_XXX macros depending on configuration.
`f_rng`	The RNG function to use. This must not be NULL.

Parameter	Description
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX code on failure.

3.11.7.4 void mbedtls_ecdsa_init (mbedtls_ecdsa_context * ctx)

Parameters:

Parameter	Description
`ctx`	The ECDSA context to initialize. This must not be NULL.

3.11.7.5 int mbedtls_ecdsa_read_signature (mbedtls_ecdsa_context * ctx, const unsigned char * hash, size_t hlen, const unsigned char * sig, size_t slen)

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.4, step 3.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDSA context to use. This must be initialized and have a group and public key bound to it.
`hash`	The message hash that was signed. This must be a readable buffer of length size bytes.
`hlen`	The size of the hash hash.
`sig`	The signature to read and verify. This must be a readable buffer of length slen bytes.
`slen`	The size of sig in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if signature is invalid.

MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH if there is a valid signature in sig , but its length is less than siglen.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_ERR_MPI_XXX error code on failure for any other reason.

3.11.7.6 int mbedtls_ecdsa_read_signature_restartable (mbedtls_ecdsa_context * ctx, const unsigned char * hash, size_t hlen, const unsigned char * sig, size_t slen, mbedtls_ecdsa_restart_ctx * rs_ctx)

See also:

mbedtls_ecdsa_read_signature()

This function is like **mbedtls_ecdsa_read_signature()** but it can return early and restart according to the limit set with mbedtls_ecp_set_max_ops() to reduce blocking.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to use. This must be initialized and have a group and public key bound to it.
`hash`	The message hash that was signed. This must be a readable buffer of length size bytes.
`hlen`	The size of the hash hash.
`sig`	The signature to read and verify. This must be a readable buffer of length slen bytes.
`slen`	The size of sig in bytes.
`rs_ctx`	The restart context to use. This may be NULL to disable restarting. If it is not NULL , it must point to an initialized restart context.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if signature is invalid.

MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH if there is a valid signature in sig , but its length is less than siglen.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another MBEDTLS_ERR_ECP_XXX or MBEDTLS_ERR_MPI_XXX error code on failure for any other reason.

3.11.7.7 int mbedtls_ecdsa_sign (mbedtls_ecp_group * grp, mbedtls_mpi * r, mbedtls_mpi * s, const mbedtls_mpi * d, const unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

The deterministic version implemented in mbedtls_ecdsa_sign_det() is usually preferred.

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.3, step 5.

See also:

ecp.h

Parameters:

Parameter	Description
`grp`	The context for the elliptic curve to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`r`	The MPI context in which to store the first part the signature. This must be initialized.
`s`	The MPI context in which to store the second part the signature. This must be initialized.
`d`	The private signing key. This must be initialized.
`buf`	The content to be signed. This is usually the hash of the original data to be signed. This must be a readable buffer of length blen bytes. It may be NULL if blen is zero.
`blen`	The length of buf in bytes.
`f_rng`	The RNG function. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context parameter.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.11.7.8 int mbedtls_ecdsa_verify (mbedtls_ecp_group * grp, const unsigned char * buf, size_t blen, const mbedtls_ecp_point * Q, const mbedtls_mpi * r, const mbedtls_mpi * s)

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.4, step 3.

See also:

ecp.h

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`buf`	The hashed content that was signed. This must be a readable buffer of length blen bytes. It may be NULL if blen is zero.
`blen`	The length of buf in bytes.
`Q`	The public key to use for verification. This must be initialized and setup.
`r`	The first integer of the signature. This must be initialized.
`s`	The second integer of the signature. This must be initialized.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if the signature is invalid.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure for any other reason.

3.11.7.9 int mbedtls_ecdsa_write_signature (mbedtls_ecdsa_context * ctx, mbedtls_md_type_t md_alg, const unsigned char * hash, size_t hlen, unsigned char * sig, size_t * slen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

It is not thread-safe to use the same context in multiple threads.

The deterministic version is used if #MBEDTLS_ECDSA_DETERMINISTIC is defined. For more information, see *RFC-6979 Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).*

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in *SECG SEC1 Elliptic Curve Cryptography*, section 4.1.3, step 5.

See also:

ecp.h

Parameters:

Parameter	Description
`ctx`	The ECDSA context to use. This must be initialized and have a group and private key bound to it, for example via **mbedtls_ecdsa_genkey()** or **mbedtls_ecdsa_from_keypair()**.
`md_alg`	The message digest that was used to hash the message.
`hash`	The message hash to be signed. This must be a readable buffer of length blen bytes.
`hlen`	The length of the hash hash in bytes.
`sig`	The buffer to which to write the signature. This must be a writable buffer of length at least twice as large as the size of the curve used, plus 9. For example, 73 bytes if a 256-bit curve is used. A buffer length of **MBEDTLS_ECDSA_MAX_LEN** is always safe.
`slen`	The address at which to store the actual length of the signature written. Must not be NULL.
`f_rng`	The RNG function. This must not be NULL if #MBEDTLS_ECDSA_DETERMINISTIC is unset. Otherwise, it is unused and may be set to NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't use a context.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX , MBEDTLS_ERR_MPI_XXX or MBEDTLS_ERR_ASN1_XXX error code on failure.

3.11.7.10 int mbedtls_ecdsa_write_signature_restartable (mbedtls_ecdsa_context * ctx, mbedtls_md_type_t md_alg, const unsigned char * hash, size_t hlen, unsigned char * sig, size_t * slen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, mbedtls_ecdsa_restart_ctx * rs_ctx)

See also:

mbedtls_ecdsa_write_signature()

This function is like **mbedtls_ecdsa_write_signature()** but it can return early and restart according to the limit set with mbedtls_ecp_set_max_ops() to reduce blocking.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to use. This must be initialized and have a group and private key bound to it, for example via **mbedtls_ecdsa_genkey()** or **mbedtls_ecdsa_from_keypair()**.
`md_alg`	The message digest that was used to hash the message.
`hash`	The message hash to be signed. This must be a readable buffer of length blen bytes.
`hlen`	The length of the hash hash in bytes.
`sig`	The buffer to which to write the signature. This must be a writable buffer of length at least twice as large as the size of the curve used, plus 9. For example, 73 bytes if a 256-bit curve is used. A buffer length of **MBEDTLS_ECDSA_MAX_LEN** is always safe.
`slen`	The address at which to store the actual length of the signature written. Must not be NULL.
`f_rng`	The RNG function. This must not be NULL if #MBEDTLS_ECDSA_DETERMINISTIC is unset. Otherwise, it is unused and may be set to NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't use a context.
`rs_ctx`	The restart context to use. This may be NULL to disable restarting. If it is not NULL, it must point to an initialized restart context.

Returns:

0 on success.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX or MBEDTLS_ERR_ASN1_XXX error code on failure.

3.12 ecp.h File reference

This file provides an API for Elliptic Curves over GF(P) (ECP).

```
#include "config.h"

#include "bignum.h"
```

3.12.1 Data structures

- struct **mbedtls_ecp_curve_info**
- struct **mbedtls_ecp_point**

  The ECP point structure, in Jacobian coordinates.
- struct **mbedtls_ecp_group**

  The ECP group structure.
- struct **mbedtls_ecp_keypair**

  The ECP key-pair structure.

3.12.2 Macros

- #define **MBEDTLS_ERR_ECP_BAD_INPUT_DATA** -0x4F80
- #define **MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL** -0x4F00
- #define **MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE** -0x4E80
- #define **MBEDTLS_ERR_ECP_VERIFY_FAILED** -0x4E00
- #define **MBEDTLS_ERR_ECP_ALLOC_FAILED** -0x4D80
- #define **MBEDTLS_ERR_ECP_RANDOM_FAILED** -0x4D00
- #define **MBEDTLS_ERR_ECP_INVALID_KEY** -0x4C80
- #define **MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH** -0x4C00
- #define **MBEDTLS_ERR_ECP_HW_ACCEL_FAILED** -0x4B80
- #define **MBEDTLS_ERR_ECP_IN_PROGRESS** -0x4B00
- #define **MBEDTLS_ECP_DP_MAX** 12
- #define MBEDTLS_ECP_BUDGET(ops) /*no-op; for compatibility */
- #define **MBEDTLS_ECP_PF_UNCOMPRESSED** 0
- #define **MBEDTLS_ECP_PF_COMPRESSED** 1
- #define **MBEDTLS_ECP_TLS_NAMED_CURVE** 3

3.12.3 Module settings

The configuration options you can set for this module are in this section. Either change them in config.h, or define them using the compiler command line.

- #define **MBEDTLS_ECP_MAX_BITS** 521
- #define MBEDTLS_ECP_MAX_BYTES ((**MBEDTLS_ECP_MAX_BITS** + 7) / 8)
- #define MBEDTLS_ECP_MAX_PT_LEN (2 *MBEDTLS_ECP_MAX_BYTES + 1)
- #define **MBEDTLS_ECP_WINDOW_SIZE** 6
- #define **MBEDTLS_ECP_FIXED_POINT_OPTIM** 1

3.12.3.1 Typedefs

- typedef struct **mbedtls_ecp_curve_info mbedtls_ecp_curve_info**
- typedef struct **mbedtls_ecp_point mbedtls_ecp_point**

  The ECP point structure, in Jacobian coordinates.
- typedef struct **mbedtls_ecp_group mbedtls_ecp_group**

  The ECP group structure.
- typedef void mbedtls_ecp_restart_ctx
- typedef struct **mbedtls_ecp_keypair mbedtls_ecp_keypair**

  The ECP key-pair structure.

3.12.3.2 Enumerations

- enum **mbedtls_ecp_group_id** { **MBEDTLS_ECP_DP_NONE** = 0, **MBEDTLS_ECP_DP_SECP192R1**, **MBEDTLS_ECP_DP_SECP224R1**, **MBEDTLS_ECP_DP_SECP256R1**, **MBEDTLS_ECP_DP_SECP384R1**, **MBEDTLS_ECP_DP_SECP521R1**, **MBEDTLS_ECP_DP_BP256R1**, **MBEDTLS_ECP_DP_BP384R1**, **MBEDTLS_ECP_DP_BP512R1**, **MBEDTLS_ECP_DP_CURVE25519**, **MBEDTLS_ECP_DP_SECP192K1**, **MBEDTLS_ECP_DP_SECP224K1**, **MBEDTLS_ECP_DP_SECP256K1**, **MBEDTLS_ECP_DP_CURVE448** }

3.12.3.3 Functions

- const **mbedtls_ecp_curve_info** ***mbedtls_ecp_curve_list** (void)

  This function retrieves the information defined in **mbedtls_ecp_curve_info()** for all supported curves in order of preference.
- const **mbedtls_ecp_group_id** ***mbedtls_ecp_grp_id_list** (void)

  This function retrieves the list of internal group identifiers of all supported curves in the order of preference.

- const **mbedtls_ecp_curve_info** *__mbedtls_ecp_curve_info_from_grp_id__ (**mbedtls_ecp_group_id** grp_id)

  This function retrieves curve information from an internal group identifier.
- const **mbedtls_ecp_curve_info** *__mbedtls_ecp_curve_info_from_tls_id__ (uint16_t tls_id)

  This function retrieves curve information from a TLS NamedCurve value.
- const **mbedtls_ecp_curve_info** *__mbedtls_ecp_curve_info_from_name__ (const char *name)

  This function retrieves curve information from a human-readable name.
- void **mbedtls_ecp_point_init** (**mbedtls_ecp_point** *pt)

  This function initializes a point as zero.
- void **mbedtls_ecp_group_init** (**mbedtls_ecp_group** *grp)

  This function initializes an ECP group context without loading any domain parameters.
- void **mbedtls_ecp_keypair_init** (**mbedtls_ecp_keypair** *key)

  This function initializes a key pair as an invalid one.
- void **mbedtls_ecp_point_free** (**mbedtls_ecp_point** *pt)

  This function frees the components of a point.
- void **mbedtls_ecp_group_free** (**mbedtls_ecp_group** *grp)

  This function frees the components of an ECP group.
- void **mbedtls_ecp_keypair_free** (**mbedtls_ecp_keypair** *key)

  This function frees the components of a key pair.
- int **mbedtls_ecp_copy** (**mbedtls_ecp_point** *P, const **mbedtls_ecp_point** *Q)

  This function copies the contents of point Q into point P.
- int **mbedtls_ecp_group_copy** (**mbedtls_ecp_group** *dst, const **mbedtls_ecp_group** *src)

  This function copies the contents of group src into group dst.
- int **mbedtls_ecp_set_zero** (**mbedtls_ecp_point** *pt)

  This function sets a point to the point at infinity.
- int **mbedtls_ecp_is_zero** (**mbedtls_ecp_point** *pt)

  This function checks if a point is the point at infinity.
- int **mbedtls_ecp_point_cmp** (const **mbedtls_ecp_point** *P, const **mbedtls_ecp_point** *Q)

  This function compares two points.
- int **mbedtls_ecp_point_read_string** (**mbedtls_ecp_point** *P, int radix, const char *x, const char *y)

This function imports a non-zero point from two ASCII strings.

- int **mbedtls_ecp_point_write_binary** (const **mbedtls_ecp_group** *grp, const **mbedtls_ecp_point** *P, int format, size_t *olen, unsigned char *buf, size_t buflen)

  This function exports a point into unsigned binary data.

- int **mbedtls_ecp_point_read_binary** (const **mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *P, const unsigned char *buf, size_t ilen)

  This function imports a point from unsigned binary data.

- int **mbedtls_ecp_tls_read_point** (const **mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *pt, const unsigned char **buf, size_t len)

  This function imports a point from a TLS ECPoint record.

- int **mbedtls_ecp_tls_write_point** (const **mbedtls_ecp_group** *grp, const **mbedtls_ecp_point** *pt, int format, size_t *olen, unsigned char *buf, size_t blen)

  This function exports a point as a TLS ECPoint record defined in *RFC-4492*, Section 5.4.

- int **mbedtls_ecp_group_load** (**mbedtls_ecp_group** *grp, **mbedtls_ecp_group_id** id)

  This function sets up an ECP group context from a standardized set of domain parameters.

- int **mbedtls_ecp_tls_read_group** (**mbedtls_ecp_group** *grp, const unsigned char **buf, size_t len)

  This function sets up an ECP group context from a TLS ECParameters record as defined in *RFC-4492*, Section 5.4.

- int **mbedtls_ecp_tls_read_group_id** (**mbedtls_ecp_group_id** *grp, const unsigned char **buf, size_t len)

  This function extracts an elliptic curve group ID from a TLS ECParameters record as defined in *RFC-4492*, Section 5.4.

- int **mbedtls_ecp_tls_write_group** (const **mbedtls_ecp_group** *grp, size_t *olen, unsigned char *buf, size_t blen)

  This function exports an elliptic curve as a TLS ECParameters record as defined in *RFC-4492*, Section 5.4.

- int **mbedtls_ecp_mul** (**mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *R, const mbedtls_mpi *m, const **mbedtls_ecp_point** *P, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function performs a scalar multiplication of a point by an integer: R = m *P.

- int **mbedtls_ecp_mul_restartable** (**mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *R, const mbedtls_mpi *m, const **mbedtls_ecp_point** *P, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, mbedtls_ecp_restart_ctx *rs_ctx)

  This function performs multiplication of a point by an integer: R = m *P in a restartable way.

- int **mbedtls_ecp_muladd** (**mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *R, const mbedtls_mpi *m, const **mbedtls_ecp_point** *P, const mbedtls_mpi *n, const **mbedtls_ecp_point** *Q)

  This function performs multiplication and addition of two points by integers: R = m *P + n *Q.

- int **mbedtls_ecp_muladd_restartable** (**mbedtls_ecp_group** *grp, **mbedtls_ecp_point** *R, const mbedtls_mpi *m, const **mbedtls_ecp_point** *P, const mbedtls_mpi *n, const **mbedtls_ecp_point** *Q, mbedtls_ecp_restart_ctx *rs_ctx)

  This function performs multiplication and addition of two points by integers: R = m *P + n *Q in a restartable way.

- int **mbedtls_ecp_check_pubkey** (const **mbedtls_ecp_group** *grp, const **mbedtls_ecp_point** *pt)

  This function checks that a point is a valid public key on this curve.

- int **mbedtls_ecp_check_privkey** (const **mbedtls_ecp_group** *grp, const mbedtls_mpi *d)

  This function checks that an mbedtls_mpi is a valid private key for this curve.

- int **mbedtls_ecp_gen_privkey** (const **mbedtls_ecp_group** *grp, mbedtls_mpi *d, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a private key.

- int **mbedtls_ecp_gen_keypair_base** (**mbedtls_ecp_group** *grp, const **mbedtls_ecp_point** *G, mbedtls_mpi *d, **mbedtls_ecp_point** *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a keypair with a configurable base point.

- int **mbedtls_ecp_gen_keypair** (**mbedtls_ecp_group** *grp, mbedtls_mpi *d, **mbedtls_ecp_point** *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECP keypair.

- int **mbedtls_ecp_gen_key** (**mbedtls_ecp_group_id** grp_id, **mbedtls_ecp_keypair** *key, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECP key.

- int **mbedtls_ecp_check_pub_priv** (const **mbedtls_ecp_keypair** *pub, const **mbedtls_ecp_keypair** *prv)

  This function checks that the keypair objects pub and prv have the same group and the same public point, and that the private key in prv is consistent with the public key.

3.12.3.4 Detailed Description

The use of ECP in cryptography and TLS is defined in *SECG SEC1 Elliptic Curve Cryptography* and *RFC-4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*.

RFC-2409 The Internet Key Exchange (IKE) defines ECP group types.

3.12.3.5 Macro Definition Documentation

3.12.3.5.1 #define MBEDTLS_ECP_DP_MAX 12

The number of supported curves, plus one for **MBEDTLS_ECP_DP_NONE**.

Montgomery curves are currently excluded.

3.12.3.5.2 #define MBEDTLS_ECP_FIXED_POINT_OPTIM 1

Enable fixed-point speed-up.

3.12.3.5.3 #define MBEDTLS_ECP_MAX_BITS 521

The maximum size of the groups, that is, of N and P. The maximum size of groups, in bits.

3.12.3.5.4 #define MBEDTLS_ECP_PF_COMPRESSED 1

Compressed point format.

3.12.3.5.5 #define MBEDTLS_ECP_PF_UNCOMPRESSED 0

Uncompressed point format.

3.12.3.5.6 #define MBEDTLS_ECP_TLS_NAMED_CURVE 3

The named_curve of ECCurveType.

3.12.3.5.7 #define MBEDTLS_ECP_WINDOW_SIZE 6

The maximum window size used.

3.12.3.5.8 #define MBEDTLS_ERR_ECP_ALLOC_FAILED -0x4D80

Memory allocation failed.

3.12.3.5.9 #define MBEDTLS_ERR_ECP_BAD_INPUT_DATA -0x4F80

Bad input parameters to function.

3.12.3.5.10 #define MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL -0x4F00

The buffer is too small to write to.

3.12.3.5.11 #define MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE -0x4E80

The requested feature is not available, for example, the requested curve is not supported.

3.12.3.5.12 #define MBEDTLS_ERR_ECP_HW_ACCEL_FAILED -0x4B80

The ECP hardware accelerator failed.

3.12.3.5.13 #define MBEDTLS_ERR_ECP_IN_PROGRESS -0x4B00

Operation in progress, call again with the same parameters to continue.

3.12.3.5.14 #define MBEDTLS_ERR_ECP_INVALID_KEY -0x4C80

Invalid private or public key.

3.12.3.5.15 #define MBEDTLS_ERR_ECP_RANDOM_FAILED -0x4D00

Generation of random value, such as ephemeral key, failed.

3.12.3.5.16 #define MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH -0x4C00

The buffer contains a valid signature followed by more data.

3.12.3.5.17 #define MBEDTLS_ERR_ECP_VERIFY_FAILED -0x4E00

The signature is not valid.

3.12.3.6 Typedef Documentation

3.12.3.6.1 typedef struct mbedtls_ecp_curve_info mbedtls_ecp_curve_info

Curve information, for use by other modules.

3.12.3.6.2 typedef struct mbedtls_ecp_group mbedtls_ecp_group

We consider two types of curve equations:

- Short Weierstrass: y^2 = x^3 + A x + B mod P (*SECG SEC1* + *RFC-4492*)
- Montgomery: y^2 = x^3 + A x^2 + x mod P (*Curve25519*, *Curve448*)

In both cases, the generator (G) for a prime-order subgroup is fixed.

For Short Weierstrass, this subgroup is the whole curve, and its cardinality is denoted by N. Our code requires that N is an odd prime as **mbedtls_ecp_mul()** requires an odd number, and **mbedtls_ecdsa_sign()** requires that it is prime for blinding purposes.

For Montgomery curves, we do not store A , but (A + 2) / 4 , which is the quantity used in the formulas. Additionally, nbits is not the size of N but the required size for private keys.

If modp is NULL, reduction modulo P is done using a generic algorithm. Otherwise, modp must point to a function that takes an mbedtls_mpi in the range of 0..2^(2*pbits)-1 , and transforms it in-place to an integer which is congruent mod P to the given MPI, and is close enough to pbits in size, so that it may be efficiently brought in the 0..P-1 range by a few additions or subtractions. Therefore, it is only an approximative modular reduction. It must return 0 on success and non-zero on failure.

Alternative implementations must keep the group IDs distinct. If two group structures have the same ID, then they must be identical.

3.12.3.6.3 typedef struct mbedtls_ecp_keypair mbedtls_ecp_keypair

A generic key-pair that may be used for ECDSA and fixed ECDH, for example.

Members are deliberately in the same order as in the **mbedtls_ecdsa_context** structure.

3.12.3.6.4 typedef struct mbedtls_ecp_point mbedtls_ecp_point

All functions expect and return points satisfying the following condition: Z == 0 or Z == 1. Other values of Z are used only by internal functions. The point is zero, or "at infinity", if Z == 0. Otherwise, X and Y are its standard (affine) coordinates.

3.12.3.7 Enumeration Type Documentation

3.12.3.7.1 enum mbedtls_ecp_group_id

Domain-parameter identifiers: curve, subgroup, and generator.

Only curves over prime fields are supported.

This library does not support validation of arbitrary domain parameters. Therefore, only standardized domain parameters from trusted sources should be used. See **mbedtls_ecp_group_load()**

Enumerator:

Enum	Description
`MBEDTLS_ECP_DP_NONE`	Curve not defined.
`MBEDTLS_ECP_DP_SECP192R1`	Domain parameters for the 192-bit curve defined by *FIPS 186-4* and *SECG SEC1*.
`MBEDTLS_ECP_DP_SECP224R1`	Domain parameters for the 224-bit curve defined by *FIPS 186-4* and *SECG SEC1*.

Enum	Description
`MBEDTLS_ECP_DP_SECP256R1`	Domain parameters for the 256-bit curve defined by *FIPS 186-4* and *SECG SEC1*.
`MBEDTLS_ECP_DP_SECP384R1`	Domain parameters for the 384-bit curve defined by *FIPS 186-4* and *SECG SEC1*.
`MBEDTLS_ECP_DP_SECP521R1`	Domain parameters for the 521-bit curve defined by *FIPS 186-4* and *SECG SEC1*.
`MBEDTLS_ECP_DP_BP256R1`	Domain parameters for 256-bit Brainpool curve.
`MBEDTLS_ECP_DP_BP384R1`	Domain parameters for 384-bit Brainpool curve.
`MBEDTLS_ECP_DP_BP512R1`	Domain parameters for 512-bit Brainpool curve.
`MBEDTLS_ECP_DP_CURVE25519`	Domain parameters for Curve25519.
`MBEDTLS_ECP_DP_SECP192K1`	Domain parameters for 192-bit "Koblitz" curve.
`MBEDTLS_ECP_DP_SECP224K1`	Domain parameters for 224-bit "Koblitz" curve.
`MBEDTLS_ECP_DP_SECP256K1`	Domain parameters for 256-bit "Koblitz" curve.
`MBEDTLS_ECP_DP_CURVE448`	Domain parameters for *Curve448*.

3.12.3.8 Function Documentation

3.12.3.8.1 int mbedtls_ecp_check_privkey (const mbedtls_ecp_group * grp, const mbedtls_mpi * d)

This function uses bare components rather than an **mbedtls_ecp_keypair** structure to ease use with other structures, such as **mbedtls_ecdh_context** or **mbedtls_ecdsa_context**.

Parameters:

Parameter	Description
`grp`	The ECP group the private key should belong to. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`d`	The integer to check. This must be initialized.

Returns:

0 if the point is a valid private key.

MBEDTLS_ERR_ECP_INVALID_KEY if the point is not a valid private key for the given curve.

Another negative error code on other kinds of failure.

3.12.3.8.2 int mbedtls_ecp_check_pub_priv (const mbedtls_ecp_keypair * pub, const mbedtls_ecp_keypair * prv)

Parameters:

Parameter	Description
`pub`	The keypair structure holding the public key. This must be initialized. If it contains a private key, that part is ignored.
`prv`	The keypair structure holding the full keypair. This must be initialized.

Returns:

0 on success, meaning that the keys are valid and match.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if the keys are invalid or do not match.

An MBEDTLS_ERR_ECP_XXX or an MBEDTLS_ERR_MPI_XXX error code on calculation failure.

3.12.3.8.3 int mbedtls_ecp_check_pubkey (const mbedtls_ecp_group * grp, const mbedtls_ecp_point * pt)

It only checks that the point is non-zero, has valid coordinates and lies on the curve. It does not verify that it is indeed a multiple of G. This additional check is computationally more expensive, is not required by standards, and should not be necessary if the group used has a small cofactor. In particular, it is useless for the NIST groups which all have a cofactor of 1.

This function uses bare components rather than an **mbedtls_ecp_keypair** structure, to ease use with other structures, such as **mbedtls_ecdh_context** or **mbedtls_ecdsa_context**.

Parameters:

Parameter	Description
`grp`	The ECP group the point should belong to. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`pt`	The point to check. This must be initialized.

Returns:

0 if the point is a valid public key.

MBEDTLS_ERR_ECP_INVALID_KEY if the point is not a valid public key for the given curve.

Another negative error code on other kinds of failure.

3.12.3.8.4 int mbedtls_ecp_copy (mbedtls_ecp_point * P, const mbedtls_ecp_point * Q)

Parameters:

Parameter	Description
P	The destination point. This must be initialized.
Q	The source point. This must be initialized.

Returns:

0 on success.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

Another negative error code for other kinds of failure.

3.12.3.8.5 const mbedtls_ecp_curve_info*mbedtls_ecp_curve_info_from_grp_id (mbedtls_ecp_group_id grp_id)

Parameters:

Parameter	Description
grp_id	An MBEDTLS_ECP_DP_XXX value.

Returns:

The associated curve information on success.

NULL on failure.

3.12.3.8.6 const mbedtls_ecp_curve_info*mbedtls_ecp_curve_info_from_name (const char * name)

Parameters:

Parameter	Description
name	The human-readable name.

Returns:

The associated curve information on success.

NULL on failure.

3.12.3.8.7 const mbedtls_ecp_curve_info*mbedtls_ecp_curve_info_from_tls_id (uint16_t tls_id)

Parameters:

Parameter	Description
`tls_id`	An MBEDTLS_ECP_DP_XXX value.

Returns:

The associated curve information on success.

NULL on failure.

3.12.3.8.8 const mbedtls_ecp_curve_info*mbedtls_ecp_curve_list (void)

Returns:

A statically allocated array. The last entry is 0.

3.12.3.8.9 int mbedtls_ecp_gen_key (mbedtls_ecp_group_id grp_id, mbedtls_ecp_keypair * key, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Parameters:

Parameter	Description
`grp_id`	The ECP group identifier.
`key`	The destination key. This must be initialized.
`f_rng`	The RNG function to use. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.12.3.8.10 int mbedtls_ecp_gen_keypair (mbedtls_ecp_group * grp, mbedtls_mpi * d, mbedtls_ecp_point * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function uses bare components rather than an **mbedtls_ecp_keypair** structure to ease use with other structures, such as **mbedtls_ecdh_context** or **mbedtls_ecdsa_context**.

Parameters:

Parameter	Description
`grp`	The ECP group to generate a key pair for. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`d`	The destination MPI (secret part). This must be initialized.
`Q`	The destination point (public part). This must be initialized.
`f_rng`	The RNG function. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.12.3.8.11 int mbedtls_ecp_gen_keypair_base (mbedtls_ecp_group * grp, const mbedtls_ecp_point * G, mbedtls_mpi * d, mbedtls_ecp_point * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function uses bare components rather than an **mbedtls_ecp_keypair** structure to ease use with other structures, such as **mbedtls_ecdh_context** or **mbedtls_ecdsa_context**.

Parameters:

Parameter	Description
`grp`	The ECP group to generate a key pair for. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`G`	The base point to use. This must be initialized and belong to grp. It replaces the default base point grp->G used by **mbedtls_ecp_gen_keypair()**.
`d`	The destination MPI (secret part). This must be initialized.
`Q`	The destination point (public part). This must be initialized.
`f_rng`	The RNG function. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.12.3.8.12 int mbedtls_ecp_gen_privkey (const mbedtls_ecp_group * grp, mbedtls_mpi * d, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Parameters:

Parameter	Description
`grp`	The ECP group to generate a private key for. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`d`	The destination MPI (secret part). This must be initialized.
`f_rng`	The RNG function. This must not be NULL.
`p_rng`	The RNG parameter to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.

Returns:

0 on success.

An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

3.12.3.8.13 int mbedtls_ecp_group_copy (mbedtls_ecp_group * dst, const mbedtls_ecp_group * src)

Parameters:

Parameter	Description
`dst`	The destination group. This must be initialized.
`src`	The source group. This must be initialized.

Returns:

0 on success.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

Another negative error code on other kinds of failure.

3.12.3.8.14 void mbedtls_ecp_group_free (mbedtls_ecp_group * grp)

Parameters:

Parameter	Description
`grp`	The group to free. This may be NULL , in which case this function returns immediately. If it is not NULL , it must point to an initialized ECP group.

3.12.3.8.15 void mbedtls_ecp_group_init (mbedtls_ecp_group * grp)

After this function is called, domain parameters for various ECP groups can be loaded through the **mbedtls_ecp_group_load()** or **mbedtls_ecp_tls_read_group()** functions.

3.12.3.8.16 int mbedtls_ecp_group_load (mbedtls_ecp_group * grp, mbedtls_ecp_group_id id)

The index should be a value of the NamedCurve enum, as defined in *RFC-4492 Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*, usually in the form of an MBEDTLS_ECP_DP_XXX macro.

Parameters:

Parameter	Description
`grp`	The group context to setup. This must be initialized.
`id`	The identifier of the domain parameter set to load.

Returns:

0 on success.

MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if id doesn't correspond to a known group.

Another negative error code on other kinds of failure.

3.12.3.8.17 const mbedtls_ecp_group_id*mbedtls_ecp_grp_id_list (void)

Returns:

A statically allocated array, terminated with MBEDTLS_ECP_DP_NONE.

3.12.3.8.18 int mbedtls_ecp_is_zero (mbedtls_ecp_point * pt)

Parameters:

Parameter	Description
`pt`	The point to test. This must be initialized.

Returns:

1 if the point is zero.

0 if the point is non-zero.

A negative error code on failure.

3.12.3.8.19 void mbedtls_ecp_keypair_free (mbedtls_ecp_keypair * key)

Parameters:

Parameter	Description
key	The key pair to free. This may be NULL , in which case this function returns immediately. If it is not NULL , it must point to an initialized ECP key pair.

3.12.3.8.20 void mbedtls_ecp_keypair_init (mbedtls_ecp_keypair * key)

Parameters:

Parameter	Description
key	The key pair to initialize.

3.12.3.8.21 int mbedtls_ecp_mul (mbedtls_ecp_group * grp, mbedtls_ecp_point * R, const mbedtls_mpi * m, const mbedtls_ecp_point * P, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

It is not thread-safe to use same group in multiple threads.

To prevent timing attacks, this function executes the exact same sequence of base-field operations for any valid m. It avoids any if-branch or array index depending on the value of m.

If f_rng is not NULL, it is used to randomize intermediate results to prevent potential timing attacks targeting these results. We recommend always providing a non-NULL f_rng. The overhead is negligible.

Parameters:

Parameter	Description
grp	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
R	The point in which to store the result of the calculation. This must be initialized.
m	The integer by which to multiply. This must be initialized.
P	The point to multiply. This must be initialized.
f_rng	The RNG function. This may be NULL if randomization of intermediate results isn't desired (discouraged).
p_rng	The RNG context to be passed to p_rng.

Returns:

0 on success.

MBEDTLS_ERR_ECP_INVALID_KEY if m is not a valid private key, or P is not a valid public key.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

Another negative error code on other kinds of failure.

3.12.3.8.22 int mbedtls_ecp_mul_restartable (mbedtls_ecp_group * grp, mbedtls_ecp_point * R, const mbedtls_mpi * m, const mbedtls_ecp_point * P, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, mbedtls_ecp_restart_ctx * rs_ctx)

See also:

mbedtls_ecp_mul()

This function does the same as **mbedtls_ecp_mul()** , but it can return early and restart according to the limit set with mbedtls_ecp_set_max_ops() to reduce blocking.

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`R`	The point in which to store the result of the calculation. This must be initialized.
`m`	The integer by which to multiply. This must be initialized.
`P`	The point to multiply. This must be initialized.
`f_rng`	The RNG function. This may be NULL if randomization of intermediate results isn't desired (discouraged).
`p_rng`	The RNG context to be passed to p_rng.
`rs_ctx`	The restart context (NULL disables restart).

Returns:

0 on success.

MBEDTLS_ERR_ECP_INVALID_KEY if m is not a valid private key, or P is not a valid public key.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another negative error code on other kinds of failure.

3.12.3.8.23 int mbedtls_ecp_muladd (mbedtls_ecp_group * grp, mbedtls_ecp_point * R, const mbedtls_mpi * m, const mbedtls_ecp_point * P, const mbedtls_mpi * n, const mbedtls_ecp_point * Q)

It is not thread-safe to use same group in multiple threads.

In contrast to **mbedtls_ecp_mul()**, this function does not guarantee a constant execution flow and timing.

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`R`	The point in which to store the result of the calculation. This must be initialized.
`m`	The integer by which to multiply P. This must be initialized.
`P`	The point to multiply by m. This must be initialized.
`n`	The integer by which to multiply Q. This must be initialized.
`Q`	The point to be multiplied by n. This must be initialized.

Returns:

0 on success.

MBEDTLS_ERR_ECP_INVALID_KEY if m or n are not valid private keys, or P or Q are not valid public keys.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

Another negative error code on other kinds of failure.

3.12.3.8.24 int mbedtls_ecp_muladd_restartable (mbedtls_ecp_group * grp, mbedtls_ecp_point * R, const mbedtls_mpi * m, const mbedtls_ecp_point * P, const mbedtls_mpi * n, const mbedtls_ecp_point * Q, mbedtls_ecp_restart_ctx * rs_ctx)

See also:

mbedtls_ecp_muladd()

This function works the same as **mbedtls_ecp_muladd()** , but it can return early and restart according to the limit set with mbedtls_ecp_set_max_ops() to reduce blocking.

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.

Parameter	Description
R	The point in which to store the result of the calculation. This must be initialized.
m	The integer by which to multiply P. This must be initialized.
P	The point to multiply by m. This must be initialized.
n	The integer by which to multiply Q. This must be initialized.
Q	The point to be multiplied by n. This must be initialized.
rs_ctx	The restart context (NULL disables restart).

Returns:

0 on success.

MBEDTLS_ERR_ECP_INVALID_KEY if m or n are not valid private keys, or P or Q are not valid public keys.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

MBEDTLS_ERR_ECP_IN_PROGRESS if maximum number of operations was reached: see mbedtls_ecp_set_max_ops().

Another negative error code on other kinds of failure.

3.12.3.8.25 int mbedtls_ecp_point_cmp (const mbedtls_ecp_point * P, const mbedtls_ecp_point * Q)

This assumes that the points are normalized. Otherwise, they may compare as "not equal" even if they are.

Parameters:

Parameter	Description
P	The first point to compare. This must be initialized.
Q	The second point to compare. This must be initialized.

Returns:

0 if the points are equal.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if the points are not equal.

3.12.3.8.26 void mbedtls_ecp_point_free (mbedtls_ecp_point * pt)

Parameters:

Parameter	Description
pt	The point to free.

3.12.3.8.27 void mbedtls_ecp_point_init (mbedtls_ecp_point * pt)

Parameters:

Parameter	Description
`pt`	The point to initialize.

3.12.3.8.28 int mbedtls_ecp_point_read_binary (const mbedtls_ecp_group * grp, mbedtls_ecp_point * P, const unsigned char * buf, size_t ilen)

This function does not check that the point actually belongs to the given group, see **mbedtls_ecp_check_pubkey()** for that.

Parameters:

Parameter	Description
`grp`	The group to which the point should belong. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`P`	The destination context to import the point to. This must be initialized.
`buf`	The input buffer. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input buffer buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if the input is invalid.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if the point format is not implemented.

3.12.3.8.29 int mbedtls_ecp_point_read_string (mbedtls_ecp_point * P, int radix, const char * x, const char * y)

Parameters:

Parameter	Description
`P`	The destination point. This must be initialized.
`radix`	The numeric base of the input.
`x`	The first affine coordinate, as a null-terminated string.
`y`	The second affine coordinate, as a null-terminated string.

Returns:

0 on success.

An MBEDTLS_ERR_MPI_XXX error code on failure.

3.12.3.8.30 int mbedtls_ecp_point_write_binary (const mbedtls_ecp_group * grp, const mbedtls_ecp_point * P, int format, size_t * olen, unsigned char * buf, size_t buflen)

Parameters:

Parameter	Description
`grp`	The group to which the point should belong. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`P`	The point to export. This must be initialized.
`format`	The point format. This must be either **MBEDTLS_ECP_PF_COMPRESSED** or **MBEDTLS_ECP_PF_UNCOMPRESSED**.
`olen`	The address at which to store the length of the output in bytes. This must not be NULL.
`buf`	The output buffer. This must be a writable buffer of length buflen bytes.
`buflen`	The length of the output buffer buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL if the output buffer is too small to hold the point.

Another negative error code on other kinds of failure.

3.12.3.8.31 int mbedtls_ecp_set_zero (mbedtls_ecp_point * pt)

Parameters:

Parameter	Description
`pt`	The point to set. This must be initialized.

Returns:

0 on success.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory-allocation failure.

Another negative error code on other kinds of failure.

3.12.3.8.32 int mbedtls_ecp_tls_read_group (mbedtls_ecp_group * grp, const unsigned char ** buf, size_t len)

The read pointer buf is updated to point right after the ECParameters record on exit.

Parameters:

Parameter	Description
`grp`	The group context to setup. This must be initialized.
`buf`	The address of the pointer to the start of the input buffer.
`len`	The length of the input buffer *buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid.

MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if the group is not recognized.

Another negative error code on other kinds of failure.

3.12.3.8.33 int mbedtls_ecp_tls_read_group_id (mbedtls_ecp_group_id * grp, const unsigned char ** buf, size_t len)

The read pointer buf is updated to point right after the ECParameters record on exit.

Parameters:

Parameter	Description
`grp`	The address at which to store the group id. This must not be NULL.
`buf`	The address of the pointer to the start of the input buffer.
`len`	The length of the input buffer *buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid.

MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if the group is not recognized.

Another negative error code on other kinds of failure.

3.12.3.8.34 int mbedtls_ecp_tls_read_point (const mbedtls_ecp_group * grp, mbedtls_ecp_point * pt, const unsigned char ** buf, size_t len)

On function return, *buf is updated to point immediately after the ECPoint record.

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`pt`	The destination point.
`buf`	The address of the pointer to the start of the input buffer.
`len`	The length of the buffer.

Returns:

0 on success.

An MBEDTLS_ERR_MPI_XXX error code on initialization failure.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid.

3.12.3.8.35 int mbedtls_ecp_tls_write_group (const mbedtls_ecp_group * grp, size_t * olen, unsigned char * buf, size_t blen)

Parameters:

Parameter	Description
`grp`	The ECP group to be exported. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`olen`	The address at which to store the number of bytes written. This must not be NULL.
`buf`	The buffer to write to. This must be a writable buffer of length blen bytes.
`blen`	The length of the output buffer buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL if the output buffer is too small to hold the exported group.

Another negative error code on other kinds of failure.

3.12.3.8.36 int mbedtls_ecp_tls_write_point (const mbedtls_ecp_group * grp, const mbedtls_ecp_point * pt, int format, size_t * olen, unsigned char * buf, size_t blen)

Parameters:

Parameter	Description
`grp`	The ECP group to use. This must be initialized and have group parameters set, for example through **mbedtls_ecp_group_load()**.
`pt`	The point to be exported. This must be initialized.
`format`	The point format to use. This must be either **MBEDTLS_ECP_PF_COMPRESSED** or **MBEDTLS_ECP_PF_UNCOMPRESSED**.
`olen`	The address at which to store the length in bytes of the data written.
`buf`	The target buffer. This must be a writable buffer of length blen bytes.
`blen`	The length of the target buffer buf in bytes.

Returns:

0 on success.

MBEDTLS_ERR_ECP_BAD_INPUT_DATA if the input is invalid.

MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL if the target buffer is too small to hold the exported point.

Another negative error code on other kinds of failure.

3.13 gcm.h File reference

This file contains GCM definitions and functions.

```
#include "config.h"

#include "cipher.h"

#include <stdint.h>
```

3.13.1 Data structures

- struct **mbedtls_gcm_context**

  The GCM context structure.

3.13.2 Macros

- #define MBEDTLS_GCM_ENCRYPT 1
- #define MBEDTLS_GCM_DECRYPT 0
- #define **MBEDTLS_ERR_GCM_AUTH_FAILED** -0x0012
- #define **MBEDTLS_ERR_GCM_HW_ACCEL_FAILED** -0x0013
- #define **MBEDTLS_ERR_GCM_BAD_INPUT** -0x0014

3.13.3 typedefs

- typedef struct **mbedtls_gcm_context mbedtls_gcm_context**

  The GCM context structure.

3.13.4 Functions

- void **mbedtls_gcm_init** (**mbedtls_gcm_context** *ctx)

  This function initializes the specified GCM context, to make references valid, and prepares the context for **mbedtls_gcm_setkey()** or **mbedtls_gcm_free()**.

- int **mbedtls_gcm_setkey** (**mbedtls_gcm_context** *ctx, **mbedtls_cipher_id_t** cipher, const unsigned char *key, unsigned int keybits)

  This function associates a GCM context with a cipher algorithm and a key.

- int **mbedtls_gcm_crypt_and_tag** (**mbedtls_gcm_context** *ctx, int mode, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, size_t tag_len, unsigned char *tag)

  This function performs GCM encryption or decryption of a buffer.

- int **mbedtls_gcm_auth_decrypt** (**mbedtls_gcm_context** *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *tag, size_t tag_len, const unsigned char *input, unsigned char *output)

  This function performs a GCM authenticated decryption of a buffer.
- int **mbedtls_gcm_starts** (**mbedtls_gcm_context** *ctx, int mode, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len)

  This function starts a GCM encryption or decryption operation.
- int **mbedtls_gcm_update** (**mbedtls_gcm_context** *ctx, size_t length, const unsigned char *input, unsigned char *output)

  This function feeds an input buffer into an ongoing GCM encryption or decryption operation.
- int **mbedtls_gcm_finish** (**mbedtls_gcm_context** *ctx, unsigned char *tag, size_t tag_len)

  This function finishes the GCM operation and generates the authentication tag.
- void **mbedtls_gcm_free** (**mbedtls_gcm_context** *ctx)

  This function clears a GCM context and the underlying cipher sub-context.

3.13.5 Detailed description

The Galois/Counter Mode (GCM) for 128-bit block ciphers is defined in D. McGrew, J. Viega, The Galois/Counter Mode of Operation (GCM), Natl. Inst. Stand. Technol.

For more information on GCM, see *NIST SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*.

3.13.6 Macro definition documentation

3.13.6.1 #define MBEDTLS_ERR_GCM_AUTH_FAILED -0x0012

Authenticated decryption failed.

3.13.6.2 #define MBEDTLS_ERR_GCM_BAD_INPUT -0x0014

Bad input parameters to function.

3.13.6.3 #define MBEDTLS_ERR_GCM_HW_ACCEL_FAILED -0x0013

GCM hardware accelerator failed.

3.13.7 Function documentation

3.13.7.1 int mbedtls_gcm_auth_decrypt (mbedtls_gcm_context * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * tag, size_t tag_len, const unsigned char * input, unsigned char * output)

For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 bytes behind the input buffer.

Parameters:

Parameter	Description
`ctx`	The GCM context. This must be initialized.
`length`	The length of the ciphertext to decrypt, which is also the length of the decrypted plaintext.
`iv`	The initialization vector. This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data. This must be of at least that size in bytes.
`add_len`	The length of the additional data.
`tag`	The buffer holding the tag to verify. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the tag to verify.
`input`	The buffer holding the ciphertext. If length is greater than zero, this must be a readable buffer of at least that size.
`output`	The buffer for holding the decrypted plaintext. If length is greater than zero, this must be a writable buffer of at least that size.

Returns:

0 if successful and authenticated.

MBEDTLS_ERR_GCM_AUTH_FAILED if the tag does not match.

MBEDTLS_ERR_GCM_BAD_INPUT if the lengths or pointers are not valid or a cipher-specific error code if the decryption failed.

3.13.7.2 int mbedtls_gcm_crypt_and_tag (mbedtls_gcm_context * ctx, int mode, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, size_t tag_len, unsigned char * tag)

For encryption, the output buffer can be the same as the input buffer. For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 bytes behind the input buffer.

When this function performs a decryption, it outputs the authentication tag and does not verify that the data is authentic. You should use this function to perform encryption only. For decryption, use **mbedtls_gcm_auth_decrypt()** instead.

Parameters:

Parameter	Description
`ctx`	The GCM context to use for encryption or decryption. This must be initialized.
`mode`	The operation to perform: • #MBEDTLS_GCM_ENCRYPT to perform authenticated encryption. The ciphertext is written to output and the authentication tag is written to tag. • #MBEDTLS_GCM_DECRYPT to perform decryption. The plaintext is written to output and the authentication tag is written to tag. Note that this mode is not recommended, because it does not verify the authenticity of the data. For this reason, you should use **mbedtls_gcm_auth_decrypt()** instead of calling this function in decryption mode.
`length`	The length of the input data, which is equal to the length of the output data.
`iv`	The initialization vector. This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data. This must be of at least that size in bytes.
`add_len`	The length of the additional data.
`input`	The buffer holding the input data. If length is greater than zero, this must be a readable buffer of at least that size in bytes.
`output`	The buffer for holding the output data. If length is greater than zero, this must be a writable buffer of at least that size in bytes.
`tag_len`	The length of the tag to generate.
`tag`	The buffer for holding the tag. This must be a readable buffer of at least tag_len bytes.

Returns:

0 if the encryption or decryption was performed successfully. Note that in #MBEDTLS_GCM_DECRYPT mode, this does not indicate that the data is authentic.

MBEDTLS_ERR_GCM_BAD_INPUT if the lengths or pointers are not valid or a cipher-specific error code if the encryption or decryption failed.

3.13.7.3 int mbedtls_gcm_finish (mbedtls_gcm_context * ctx, unsigned char * tag, size_t tag_len)

It wraps up the GCM stream, and generates the tag. The tag can have a maximum length of 16 bytes.

Parameters:

Parameter	Description
`ctx`	The GCM context. This must be initialized.
`tag`	The buffer for holding the tag. This must be a readable buffer of at least tag_len bytes.
`tag_len`	The length of the tag to generate. This must be at least four.

Returns:

0 on success.

MBEDTLS_ERR_GCM_BAD_INPUT on failure.

3.13.7.4 void mbedtls_gcm_free (mbedtls_gcm_context * ctx)

Parameters:

Parameter	Description
`ctx`	The GCM context to clear. If this is NULL , the call has no effect. Otherwise, this must be initialized.

3.13.7.5 void mbedtls_gcm_init (mbedtls_gcm_context * ctx)

The function does not bind the GCM context to a particular cipher, nor set the key. For this purpose, use **mbedtls_gcm_setkey()**.

Parameters:

Parameter	Description
`ctx`	The GCM context to initialize. This must not be NULL.

3.13.7.6 int mbedtls_gcm_setkey (mbedtls_gcm_context * ctx, mbedtls_cipher_id_t cipher, const unsigned char * key, unsigned int keybits)

Parameters:

Parameter	Description
`ctx`	The GCM context. This must be initialized.
`cipher`	The 128-bit block cipher to use.
`key`	The encryption key. This must be a readable buffer of at least keybits bits.
`keybits`	The key size in bits. Valid options are: 128 bits, 192 bits, or 256 bits.

Returns:

0 on success.

A cipher-specific error code on failure.

3.13.7.7 int mbedtls_gcm_starts (mbedtls_gcm_context * ctx, int mode, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len)

Parameters:

Parameter	Description
`ctx`	The GCM context. This must be initialized.
`mode`	The operation to perform: #MBEDTLS_GCM_ENCRYPT or #MBEDTLS_GCM_DECRYPT.
`iv`	The initialization vector. This must be a readable buffer of at least iv_len bytes.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data, or NULL if add_len is 0.
`add_len`	The length of the additional data. If 0 , add may be NULL.

Returns:

0 on success.

3.13.7.8 int mbedtls_gcm_update (mbedtls_gcm_context * ctx, size_t length, const unsigned char * input, unsigned char * output)

` The function expects input to be a multiple of 16 bytes. Only the last call before calling **mbedtls_gcm_finish()** can be less than 16 bytes.

For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 bytes behind the input buffer.

Parameters:

Parameter	Description
`ctx`	The GCM context. This must be initialized.
`length`	The length of the input data. This must be a multiple of 16 except in the last call before **mbedtls_gcm_finish()**.
`input`	The buffer holding the input data. If length is greater than zero, this must be a readable buffer of at least that size in bytes.
`output`	The buffer for holding the output data. If length is greater than zero, this must be a writable buffer of at least that size in bytes.

Returns:

0 on success.

MBEDTLS_ERR_GCM_BAD_INPUT on failure.

3.14 hkdf.h File reference

This file contains the HKDF interface.

```
#include "config.h"
```

```
#include "md.h"
```

3.14.1 Macros

- o HKDF Error codes#define **MBEDTLS_ERR_HKDF_BAD_INPUT_DATA** -0x5F80

3.14.2 Functions

- int **mbedtls_hkdf** (const **mbedtls_md_info_t** *md, const unsigned char *salt, size_t salt_len, const unsigned char *ikm, size_t ikm_len, const unsigned char *info, size_t info_len, unsigned char *okm, size_t okm_len)

  This is the HMAC-based Extract-and-Expand Key Derivation Function (HKDF).

- int **mbedtls_hkdf_extract** (const **mbedtls_md_info_t** *md, const unsigned char *salt, size_t salt_len, const unsigned char *ikm, size_t ikm_len, unsigned char *prk)

  Take the input keying material ikm and extract from it a fixed-length pseudorandom key prk.

- int **mbedtls_hkdf_expand** (const **mbedtls_md_info_t** *md, const unsigned char *prk, size_t prk_len, const unsigned char *info, size_t info_len, unsigned char *okm, size_t okm_len)

  Expand the supplied prk into several additional pseudorandom keys, which is the output of the HKDF.

3.14.3 Detailed description

The HMAC-based Extract-and-Expand Key Derivation Function (HKDF) is specified by *RFC-5869*.

3.14.4 Macro definition documentation

3.14.4.1 #define MBEDTLS_ERR_HKDF_BAD_INPUT_DATA -0x5F80

Bad input parameters to function.

3.14.5 Function documentation

3.14.5.1 int mbedtls_hkdf (const mbedtls_md_info_t * md, const unsigned char * salt, size_t salt_len, const unsigned char * ikm, size_t ikm_len, const unsigned char * info, size_t info_len, unsigned char * okm, size_t okm_len)

Parameters:

Parameter	Description
`md`	A hash function; md.size denotes the length of the hash function output in bytes.
`salt`	An optional salt value (a non-secret random value); if the salt is not provided, a string of all zeros of md.size length is used as the salt.
`salt_len`	The length in bytes of the optional salt.
`ikm`	The input keying material.
`ikm_len`	The length in bytes of ikm.
`info`	An optional context and application specific information string. This can be a zero-length string.
`info_len`	The length of info in bytes.
`okm`	The output keying material of okm_len bytes.
`okm_len`	The length of the output keying material in bytes. This must be less than or equal to 255 *md.size bytes.

Returns:

0 on success.

MBEDTLS_ERR_HKDF_BAD_INPUT_DATA when the parameters are invalid.

An MBEDTLS_ERR_MD_*error for errors returned from the underlying MD layer.

3.14.5.2 int mbedtls_hkdf_expand (const mbedtls_md_info_t * md, const unsigned char * prk, size_t prk_len, const unsigned char * info, size_t info_len, unsigned char * okm, size_t okm_len)

This function should only be used if the security of it has been studied and established in that particular context (eg. TLS 1.3 key schedule). For standard HKDF security guarantees use mbedtls_hkdf instead.

Parameters:

Parameter	Description
`md`	A hash function; md.size denotes the length of the hash function output in bytes.
`prk`	A pseudorandom key of at least md.size bytes. prk is usually the output from the HKDF extract step.

Parameter	Description
`prk_len`	The length in bytes of prk.
`info`	An optional context and application specific information string. This can be a zero-length string.
`info_len`	The length of info in bytes.
`okm`	The output keying material of okm_len bytes.
`okm_len`	The length of the output keying material in bytes. This must be less than or equal to 255 *md.size bytes.

Returns:

0 on success.

MBEDTLS_ERR_HKDF_BAD_INPUT_DATA when the parameters are invalid.

An MBEDTLS_ERR_MD_*error for errors returned from the underlying MD layer.

3.14.5.3 int mbedtls_hkdf_extract (const mbedtls_md_info_t * md, const unsigned char * salt, size_t salt_len, const unsigned char * ikm, size_t ikm_len, unsigned char * prk)

This function should only be used if the security of it has been studied and established in that particular context (eg. TLS 1.3 key schedule). For standard HKDF security guarantees use mbedtls_hkdf instead.

Parameters:

I/O	Parameter	Description
	`md`	A hash function; md.size denotes the length of the hash function output in bytes.
	`salt`	An optional salt value (a non-secret random value); if the salt is not provided, a string of all zeros of md.size length is used as the salt.
	`salt_len`	The length in bytes of the optional salt.
	`ikm`	The input keying material.
	`ikm_len`	The length in bytes of ikm.
out	`prk`	A pseudorandom key of at least md.size bytes.

Returns:

0 on success.

MBEDTLS_ERR_HKDF_BAD_INPUT_DATA when the parameters are invalid.

An MBEDTLS_ERR_MD_*error for errors returned from the underlying MD layer.

3.15 md.h File reference

This file contains the generic message-digest wrapper.

```
#include <stddef.h>

#include "config.h"
```

3.15.1 Data structures

- struct **mbedtls_md_context_t**

3.15.2 Macros

- #define **MBEDTLS_ERR_MD_FEATURE_UNAVAILABLE** -0x5080
- #define **MBEDTLS_ERR_MD_BAD_INPUT_DATA** -0x5100
- #define **MBEDTLS_ERR_MD_ALLOC_FAILED** -0x5180
- #define **MBEDTLS_ERR_MD_FILE_IO_ERROR** -0x5200
- #define **MBEDTLS_ERR_MD_HW_ACCEL_FAILED** -0x5280
- #define MBEDTLS_MD_MAX_SIZE 32 /*longest known is SHA256 or less */
- #define MBEDTLS_DEPRECATED

3.15.3 typedefs

- typedef struct **mbedtls_md_info_t mbedtls_md_info_t**
- typedef struct **mbedtls_md_context_t mbedtls_md_context_t**

3.15.4 Enumerations

- enum **mbedtls_md_type_t** { **MBEDTLS_MD_NONE** =0, **MBEDTLS_MD_MD2**, **MBEDTLS_MD_MD4**, **MBEDTLS_MD_MD5**, **MBEDTLS_MD_SHA1**, **MBEDTLS_MD_SHA224**, **MBEDTLS_MD_SHA256**, **MBEDTLS_MD_SHA384**, **MBEDTLS_MD_SHA512**, **MBEDTLS_MD_RIPEMD160** }

3.15.5 Supported message digests. Functions

- const int ***mbedtls_md_list** (void)

  This function returns the list of digests supported by the generic digest module.

- const **mbedtls_md_info_t** ***mbedtls_md_info_from_string** (const char *md_name)

  This function returns the message-digest information associated with the given digest name.

- const **mbedtls_md_info_t** ***mbedtls_md_info_from_type** (**mbedtls_md_type_t** md_type)

This function returns the message-digest information associated with the given digest type.

- void **mbedtls_md_init** (**mbedtls_md_context_t** *ctx)

  This function initializes a message-digest context without binding it to a particular message-digest algorithm.

- void **mbedtls_md_free** (**mbedtls_md_context_t** *ctx)

  This function clears the internal structure of ctx and frees any embedded internal structure, but does not free ctx itself.

- int **mbedtls_md_init_ctx** (**mbedtls_md_context_t** *ctx, const **mbedtls_md_info_t** *md_info) MBEDTLS_DEPRECATED

  This function selects the message digest algorithm to use, and allocates internal structures.

- int **mbedtls_md_setup** (**mbedtls_md_context_t** *ctx, const **mbedtls_md_info_t** *md_info, int hmac)

  This function selects the message digest algorithm to use, and allocates internal structures.

- int **mbedtls_md_clone** (**mbedtls_md_context_t** *dst, const **mbedtls_md_context_t** *src)

  This function clones the state of an message-digest context.

- unsigned char **mbedtls_md_get_size** (const **mbedtls_md_info_t** *md_info)

  This function extracts the message-digest size from the message-digest information structure.

- **mbedtls_md_type_t mbedtls_md_get_type** (const **mbedtls_md_info_t** *md_info)

  This function extracts the message-digest type from the message-digest information structure.

- const char ***mbedtls_md_get_name** (const **mbedtls_md_info_t** *md_info)

  This function extracts the message-digest name from the message-digest information structure.

- int **mbedtls_md_starts** (**mbedtls_md_context_t** *ctx)

  This function starts a message-digest computation.

- int **mbedtls_md_update** (**mbedtls_md_context_t** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing message-digest computation.

- int **mbedtls_md_finish** (**mbedtls_md_context_t** *ctx, unsigned char *output)

  This function finishes the digest operation, and writes the result to the output buffer.

- int **mbedtls_md** (const **mbedtls_md_info_t** *md_info, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the message-digest of a buffer, with respect to a configurable message-digest algorithm in a single call.
- int **mbedtls_md_hmac_starts** (**mbedtls_md_context_t** *ctx, const unsigned char *key, size_t keylen)

  This function sets the HMAC key and prepares to authenticate a new message.
- int **mbedtls_md_hmac_update** (**mbedtls_md_context_t** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing HMAC computation.
- int **mbedtls_md_hmac_finish** (**mbedtls_md_context_t** *ctx, unsigned char *output)

  This function finishes the HMAC operation, and writes the result to the output buffer.
- int **mbedtls_md_hmac_reset** (**mbedtls_md_context_t** *ctx)

  This function prepares to authenticate a new message with the same key as the previous HMAC operation.
- int **mbedtls_md_hmac** (const **mbedtls_md_info_t** *md_info, const unsigned char *key, size_t keylen, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the full generic HMAC on the input buffer with the provided key.
- int mbedtls_md_process (**mbedtls_md_context_t** *ctx, const unsigned char *data)

3.15.6 Detailed description

Author:

Adriaan de Jong **dejong@fox-it.com**

3.15.7 Macro definition documentation

3.15.7.1 #define MBEDTLS_ERR_MD_ALLOC_FAILED -0x5180

Failed to allocate memory.

3.15.7.2 #define MBEDTLS_ERR_MD_BAD_INPUT_DATA -0x5100

Bad input parameters to function.

3.15.7.3 #define MBEDTLS_ERR_MD_FEATURE_UNAVAILABLE -0x5080

The selected feature is not available.

3.15.7.4 #define MBEDTLS_ERR_MD_FILE_IO_ERROR -0x5200

Opening or reading of file failed.

3.15.7.5 #define MBEDTLS_ERR_MD_HW_ACCEL_FAILED -0x5280

MD hardware accelerator failed.

3.15.8 typedef documentation

3.15.8.1 typedef struct mbedtls_md_context_t mbedtls_md_context_t

The generic message-digest context.

3.15.8.2 typedef struct mbedtls_md_info_t mbedtls_md_info_t

Opaque struct defined in md_internal.h.

3.15.9 Enumeration type documentation

3.15.9.1 enum mbedtls_md_type_t

MD2, MD4, MD5 and SHA-1 are considered weak message digests and their use constitutes a security risk. We recommend considering stronger message digests instead.

Enumerator:

Enum	Description
`MBEDTLS_MD_NONE`	None.
`MBEDTLS_MD_MD2`	The MD2 message digest.
`MBEDTLS_MD_MD4`	The MD4 message digest.
`MBEDTLS_MD_MD5`	The MD5 message digest.
`MBEDTLS_MD_SHA1`	The SHA-1 message digest.
`MBEDTLS_MD_SHA224`	The SHA-224 message digest.
`MBEDTLS_MD_SHA256`	The SHA-256 message digest.
`MBEDTLS_MD_SHA384`	The SHA-384 message digest.
`MBEDTLS_MD_SHA512`	The SHA-512 message digest.
`MBEDTLS_MD_RIPEMD160`	The RIPEMD-160 message digest.

3.15.10 Function documentation

3.15.10.1 int mbedtls_md (const mbedtls_md_info_t * md_info, const unsigned char * input, size_t ilen, unsigned char * output)

The result is calculated as Output = message_digest(input buffer).

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.
`input`	The buffer holding the data.
`ilen`	The length of the input data.
`output`	The generic message-digest checksum result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.2 int mbedtls_md_clone (mbedtls_md_context_t * dst, const mbedtls_md_context_t * src)

You must call **mbedtls_md_setup()** on dst before calling this function.

The two contexts must have the same type, for example, both are SHA-256.

This function clones the message-digest state, not the HMAC state.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The context to be cloned.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.3 int mbedtls_md_finish (mbedtls_md_context_t * ctx, unsigned char * output)

Call this function after a call to **mbedtls_md_starts()**, followed by any number of calls to **mbedtls_md_update()**. Afterwards, you may either clear the context with **mbedtls_md_free()**, or call **mbedtls_md_starts()** to reuse the context for another digest operation with the same algorithm.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.
`output`	The buffer for the generic message-digest checksum result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.4 void mbedtls_md_free (mbedtls_md_context_t * ctx)

If you have called **mbedtls_md_setup()** on ctx , you must call **mbedtls_md_free()** when you are no longer using the context. Calling this function if you have previously called **mbedtls_md_init()** and nothing else is optional. You must not call this function if you have not called **mbedtls_md_init()**.

3.15.10.5 const char*mbedtls_md_get_name (const mbedtls_md_info_t * md_info)

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.

Returns:

The name of the message digest.

3.15.10.6 unsigned char mbedtls_md_get_size (const mbedtls_md_info_t * md_info)

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.

Returns:

The size of the message-digest output in bytes.

3.15.10.7 mbedtls_md_type_t **mbedtls_md_get_type (const** mbedtls_md_info_t * **md_info)**

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.

Returns:

The type of the message digest.

3.15.10.8 int mbedtls_md_hmac (const mbedtls_md_info_t * md_info, const unsigned char * key, size_t keylen, const unsigned char * input, size_t ilen, unsigned char * output)

The function allocates the context, performs the calculation, and frees the context.

The HMAC result is calculated as output = generic HMAC(hmac key, input buffer).

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.
`key`	The HMAC secret key.
`keylen`	The length of the HMAC secret key in bytes.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The generic HMAC result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.9 int mbedtls_md_hmac_finish (mbedtls_md_context_t * ctx, unsigned char * output)

Call this function after **mbedtls_md_hmac_starts()** and **mbedtls_md_hmac_update()** to get the HMAC value. Afterwards you may either call **mbedtls_md_free()** to clear the context, or call **mbedtls_md_hmac_reset()** to reuse the context with the same HMAC key.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`output`	The generic HMAC checksum result.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.10 int mbedtls_md_hmac_reset (mbedtls_md_context_t * ctx)

You may call this function after **mbedtls_md_hmac_finish()**. Afterwards call **mbedtls_md_hmac_update()** to pass the new input.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.11 int mbedtls_md_hmac_starts (mbedtls_md_context_t * ctx, const unsigned char * key, size_t keylen)

Call this function after **mbedtls_md_setup()**, to use the MD context for an HMAC calculation, then call **mbedtls_md_hmac_update()** to provide the input data, and **mbedtls_md_hmac_finish()** to get the HMAC value.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`key`	The HMAC secret key.
`keylen`	The length of the HMAC key in bytes.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.12 int mbedtls_md_hmac_update (mbedtls_md_context_t * ctx, const unsigned char * input, size_t ilen)

Call **mbedtls_md_hmac_starts()** or **mbedtls_md_hmac_reset()** before calling this function. You may call this function multiple times to pass the input piecewise. Afterwards, call **mbedtls_md_hmac_finish()**.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.13 const mbedtls_md_info_t*mbedtls_md_info_from_string (const char * md_name)

Parameters:

Parameter	Description
`md_name`	The name of the digest to search for.

Returns:

The message-digest information associated with md_name.

NULL if the associated message-digest information is not found.

3.15.10.14 const mbedtls_md_info_t*mbedtls_md_info_from_type (mbedtls_md_type_t md_type)

Parameters:

Parameter	Description
`md_type`	The type of digest to search for.

Returns:

The message-digest information associated with md_type.

NULL if the associated message-digest information is not found.

3.15.10.15 void mbedtls_md_init (mbedtls_md_context_t * ctx)

This function should always be called first. It prepares the context for **mbedtls_md_setup()** for binding it to a message-digest algorithm.

3.15.10.16 int mbedtls_md_init_ctx (mbedtls_md_context_t * ctx, const mbedtls_md_info_t * md_info)

It should be called after **mbedtls_md_init()** or **mbedtls_md_free()**. Makes it necessary to call **mbedtls_md_free()** later.

Deprecated**:**

Superseded by **mbedtls_md_setup()** in 2.0.0

Parameters:

Parameter	Description
`ctx`	The context to set up.
`md_info`	The information structure of the message-digest algorithm to use.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_MD_ALLOC_FAILED on memory-allocation failure.

3.15.10.17 const int*mbedtls_md_list (void)

Returns:

A statically allocated array of digests. Each element in the returned list is an integer belonging to the message-digest enumeration **mbedtls_md_type_t**. The last entry is 0.

3.15.10.18 int mbedtls_md_setup (mbedtls_md_context_t * ctx, const mbedtls_md_info_t * md_info, int hmac)

It should be called after **mbedtls_md_init()** or **mbedtls_md_free()**. Makes it necessary to call **mbedtls_md_free()** later.

Parameters:

Parameter	Description
`ctx`	The context to set up.
`md_info`	The information structure of the message-digest algorithm to use.

Parameter	Description
`hmac`	Defines if HMAC is used. 0: HMAC is not used (saves some memory), or non-zero: HMAC is used with this context.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

MBEDTLS_ERR_MD_ALLOC_FAILED on memory-allocation failure.

3.15.10.19 int mbedtls_md_starts (mbedtls_md_context_t * ctx)

You must call this function after setting up the context with **mbedtls_md_setup()**, and before passing data with **mbedtls_md_update()**.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.15.10.20 int mbedtls_md_update (mbedtls_md_context_t * ctx, const unsigned char * input, size_t ilen)

You must call **mbedtls_md_starts()** before calling this function. You may call this function multiple times. Afterwards, call **mbedtls_md_finish()**.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success.

MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.

3.16 nist_kw.h File reference

This file provides an API for key wrapping (KW) and key wrapping with padding (KWP) as defined in *NIST SP 800-38F*.
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf.

```
#include "config.h"
```

```
#include "cipher.h"
```

3.16.1 Data structures

- struct **mbedtls_nist_kw_context**

  The key wrapping context-type definition. The key wrapping context is passed to the APIs called.

3.16.2 Enumerations

- enum mbedtls_nist_kw_mode_t { MBEDTLS_KW_MODE_KW = 0, MBEDTLS_KW_MODE_KWP = 1 }

3.16.3 Functions

- void **mbedtls_nist_kw_init** (**mbedtls_nist_kw_context** *ctx)

  This function initializes the specified key wrapping context to make references valid and prepare the context for **mbedtls_nist_kw_setkey()** or **mbedtls_nist_kw_free()**.

- int **mbedtls_nist_kw_setkey** (**mbedtls_nist_kw_context** *ctx, **mbedtls_cipher_id_t** cipher, const unsigned char *key, unsigned int keybits, const int is_wrap)

  This function initializes the key wrapping context set in the ctx parameter and sets the encryption key.

- void **mbedtls_nist_kw_free** (**mbedtls_nist_kw_context** *ctx)

  This function releases and clears the specified key wrapping context and underlying cipher sub-context.

- int **mbedtls_nist_kw_wrap** (**mbedtls_nist_kw_context** *ctx, mbedtls_nist_kw_mode_t mode, const unsigned char *input, size_t in_len, unsigned char *output, size_t *out_len, size_t out_size)

  This function encrypts a buffer using key wrapping.

- int **mbedtls_nist_kw_unwrap** (**mbedtls_nist_kw_context** *ctx, mbedtls_nist_kw_mode_t mode, const unsigned char *input, size_t in_len, unsigned char *output, size_t *out_len, size_t out_size)

  This function decrypts a buffer using key wrapping.

3.16.4 Detailed description

Key wrapping specifies a deterministic authenticated-encryption mode of operation, according to *NIST SP 800-38F Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*. Its purpose is to protect cryptographic keys.

Its equivalent is *RFC-3394* for KW, and *RFC-5649* for KWP. **https://tools.ietf.org/html/rfc3394 https://tools.ietf.org/html/rfc5649**

3.16.5 Function documentation

3.16.5.1 void mbedtls_nist_kw_free (mbedtls_nist_kw_context * ctx)

Parameters:

Parameter	Description
`ctx`	The key wrapping context to clear.

3.16.5.2 void mbedtls_nist_kw_init (mbedtls_nist_kw_context * ctx)

Parameters:

Parameter	Description
`ctx`	The key wrapping context to initialize.

3.16.5.3 int mbedtls_nist_kw_setkey (mbedtls_nist_kw_context * ctx, mbedtls_cipher_id_t cipher, const unsigned char * key, unsigned int keybits, const int is_wrap)

Parameters:

Parameter	Description
`ctx`	The key wrapping context.
`cipher`	The 128-bit block cipher to use. Only AES is supported.
`key`	The Key Encryption Key (KEK).
`keybits`	The KEK size in bits. This must be acceptable by the cipher.
`is_wrap`	Specify whether the operation within the context is wrapping or unwrapping

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA for any invalid input.

MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE for 128-bit block ciphers which are not supported.

cipher-specific error code on failure of the underlying cipher.

3.16.5.4 int mbedtls_nist_kw_unwrap (mbedtls_nist_kw_context * ctx, mbedtls_nist_kw_mode_t mode, const unsigned char * input, size_t in_len, unsigned char * output, size_t * out_len, size_t out_size)

Parameters:

I/O	Parameter	Description
	`ctx`	The key wrapping context to use for decryption.
	`mode`	The key wrapping mode to use (MBEDTLS_KW_MODE_KW or MBEDTLS_KW_MODE_KWP)
	`input`	The buffer holding the input data.
	`in_len`	The length of the input data in bytes. The input uses units of 8 bytes called semiblocks. The input must be a multiple of semiblocks. • For KW mode: a multiple of 8 bytes between 24 and 2^57 inclusive. • For KWP mode: a multiple of 8 bytes between 16 and 2^32 inclusive.
out	`output`	The buffer holding the output data. The output buffer's minimal length is 8 bytes shorter than in_len.
out	`out_len`	The number of bytes written to the output buffer. 0 on failure. For KWP mode, the length could be up to 15 bytes shorter than in_len , depending on how much padding was added to the data.
in	`out_size`	The capacity of the output buffer.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA for invalid input length.

MBEDTLS_ERR_CIPHER_AUTH_FAILED for verification failure of the ciphertext.

cipher-specific error code on failure of the underlying cipher.

3.16.5.5 int mbedtls_nist_kw_wrap (mbedtls_nist_kw_context * ctx, mbedtls_nist_kw_mode_t mode, const unsigned char * input, size_t in_len, unsigned char * output, size_t * out_len, size_t out_size)

Parameters:

I/O	Parameter	Description
	`ctx`	The key wrapping context to use for encryption.
	`mode`	The key wrapping mode to use (MBEDTLS_KW_MODE_KW or MBEDTLS_KW_MODE_KWP)
	`input`	The buffer holding the input data.

I/O	Parameter	Description
	`in_len`	The length of the input data in bytes. The input uses units of 8 bytes called semiblocks. • For KW mode: a multiple of 8 bytes between 16 and 2^57-8 inclusive. • For KWP mode: any length between 1 and 2^32-1 inclusive.
out	`output`	The buffer holding the output data. • For KW mode: Must be at least 8 bytes larger than in_len. • For KWP mode: Must be at least 8 bytes larger rounded up to a multiple of 8 bytes for KWP (15 bytes at most).
out	`out_len`	The number of bytes written to the output buffer. 0 on failure.
in	`out_size`	The capacity of the output buffer.

Returns:

0 on success.

MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA for invalid input length.

cipher-specific error code on failure of the underlying cipher.

3.17 platform.h File reference

This file contains the definitions and functions of the Mbed TLS platform abstraction layer.

```
#include "config.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>
```

3.17.1 Data structures

- struct **mbedtls_platform_context**

  The platform context structure.

3.17.2 Macros

- #define **MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED** -0x0070
- #define **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED** -0x0072
- #define mbedtls_free free
- #define mbedtls_calloc calloc
- #define mbedtls_fprintf fprintf
- #define mbedtls_printf printf
- #define mbedtls_snprintf **MBEDTLS_PLATFORM_STD_SNPRINTF**
- #define mbedtls_exit exit
- #define MBEDTLS_EXIT_SUCCESS **MBEDTLS_PLATFORM_STD_EXIT_SUCCESS**
- #define MBEDTLS_EXIT_FAILURE **MBEDTLS_PLATFORM_STD_EXIT_FAILURE**

3.17.3 Module settings

The configuration options you can set for this module are in this section. Either change them in config.h or define them on the compiler command line.

  - #define **MBEDTLS_PLATFORM_STD_SNPRINTF** snprintf
  - #define **MBEDTLS_PLATFORM_STD_PRINTF** printf
  - #define **MBEDTLS_PLATFORM_STD_FPRINTF** fprintf
  - #define **MBEDTLS_PLATFORM_STD_CALLOC** calloc
  - #define **MBEDTLS_PLATFORM_STD_FREE** free
  - #define **MBEDTLS_PLATFORM_STD_EXIT** exit

- o #define **MBEDTLS_PLATFORM_STD_TIME** time
- o #define **MBEDTLS_PLATFORM_STD_EXIT_SUCCESS** EXIT_SUCCESS
- o #define **MBEDTLS_PLATFORM_STD_EXIT_FAILURE** EXIT_FAILURE

3.17.3.1 Typedefs

- typedef struct **mbedtls_platform_context mbedtls_platform_context**

  The platform context structure.

3.17.3.2 Functions

- int **mbedtls_platform_setup** (**mbedtls_platform_context** *ctx)

  This function performs any platform-specific initialization operations.
- void **mbedtls_platform_teardown** (**mbedtls_platform_context** *ctx)

  This function performs any platform teardown operations.

3.17.3.3 Detailed Description

The platform abstraction layer removes the need for the library to directly link to standard C library functions or operating system services, making the library easier to port and embed. Application developers and users of the library can provide their own implementations of these functions, or implementations specific to their platform, which can be statically linked to the library or dynamically configured at runtime.

3.17.3.4 Macro Definition Documentation

3.17.3.4.1 #define MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED -0x0072

The requested feature is not supported by the platform

3.17.3.4.2 #define MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED -0x0070

Hardware accelerator failed

3.17.3.4.3 #define MBEDTLS_PLATFORM_STD_CALLOC calloc

The default calloc function to use.

3.17.3.4.4 #define MBEDTLS_PLATFORM_STD_EXIT exit

The default exit function to use.

3.17.3.4.5 #define MBEDTLS_PLATFORM_STD_EXIT_FAILURE EXIT_FAILURE

The default exit value to use.

3.17.3.4.6 #define MBEDTLS_PLATFORM_STD_EXIT_SUCCESS EXIT_SUCCESS

The default exit value to use.

3.17.3.4.7 #define MBEDTLS_PLATFORM_STD_FPRINTF fprintf

The default fprintf function to use.

3.17.3.4.8 #define MBEDTLS_PLATFORM_STD_FREE free

The default free function to use.

3.17.3.4.9 #define MBEDTLS_PLATFORM_STD_PRINTF printf

The default printf function to use.

3.17.3.4.10 #define MBEDTLS_PLATFORM_STD_SNPRINTF snprintf

The default snprintf function to use.

3.17.3.4.11 #define MBEDTLS_PLATFORM_STD_TIME time

The default time function to use.

3.17.3.5 Typedef Documentation

3.17.3.5.1 typedef struct mbedtls_platform_context mbedtls_platform_context

This structure may be used to assist platform-specific setup or teardown operations.

3.17.3.6 Function Documentation

3.17.3.6.1 int mbedtls_platform_setup (mbedtls_platform_context * ctx)

This function should be called before any other library functions. Its implementation is platform-specific, and unless platform-specific code is provided, it does nothing.

The usage and necessity of this function is dependent on the platform.

Parameters:

Parameter	Description
`ctx`	The platform context.

Returns:

0 on success.

3.17.3.6.2 void mbedtls_platform_teardown (mbedtls_platform_context * ctx)

This function should be called after every other Mbed TLS module has been correctly freed using the appropriate free function. Its implementation is platform-specific, and unless platform-specific code is provided, it does nothing.

The usage and necessity of this function is dependent on the platform.

Parameters:

Parameter	Description
`ctx`	The platform context.

3.18 poly1305.h File reference

This file contains Poly1305 definitions and functions.

```
#include "config.h"
```

```
#include <stdint.h>
```

```
#include <stddef.h>
```

3.18.1 Data structures

- struct **mbedtls_poly1305_context**

3.18.2 Macros

- #define **MBEDTLS_ERR_POLY1305_BAD_INPUT_DATA** -0x0057
- #define **MBEDTLS_ERR_POLY1305_FEATURE_UNAVAILABLE** -0x0059

- #define **MBEDTLS_ERR_POLY1305_HW_ACCEL_FAILED** -0x005B

3.18.3 typedefs

- typedef struct **mbedtls_poly1305_context** mbedtls_poly1305_context

3.18.4 Functions

- void **mbedtls_poly1305_init** (**mbedtls_poly1305_context** *ctx)

  This function initializes the specified Poly1305 context.
- void **mbedtls_poly1305_free** (**mbedtls_poly1305_context** *ctx)

  This function releases and clears the specified Poly1305 context.
- int **mbedtls_poly1305_starts** (**mbedtls_poly1305_context** *ctx, const unsigned char key[32])

  This function sets the one-time authentication key.
- int **mbedtls_poly1305_update** (**mbedtls_poly1305_context** *ctx, const unsigned char *input, size_t ilen)

  This functions feeds an input buffer into an ongoing Poly1305 computation.
- int **mbedtls_poly1305_finish** (**mbedtls_poly1305_context** *ctx, unsigned char mac[16])

  This function generates the Poly1305 Message Authentication Code (MAC).
- int **mbedtls_poly1305_mac** (const unsigned char key[32], const unsigned char *input, size_t ilen, unsigned char mac[16])

  This function calculates the Poly1305 MAC of the input buffer with the provided key.

3.18.5 Detailed description

Poly1305 is a one-time message authenticator that can be used to authenticate messages. Poly1305-AES was created by Daniel Bernstein **https://cr.yp.to/mac/poly1305-20050329.pdf** The generic Poly1305 algorithm (not tied to AES) was also standardized in *RFC-7539*.

Author:

Daniel King **damaki.gh@gmail.com**

3.18.6 Macro definition documentation

3.18.6.1 #define MBEDTLS_ERR_POLY1305_BAD_INPUT_DATA -0x0057

Invalid input parameter(s).

3.18.6.2 #define MBEDTLS_ERR_POLY1305_FEATURE_UNAVAILABLE -0x0059

Feature not available. For example, s part of the API is not implemented.

3.18.6.3 #define MBEDTLS_ERR_POLY1305_HW_ACCEL_FAILED -0x005B

Poly1305 hardware accelerator failed.

3.18.7 Function documentation

3.18.7.1 int mbedtls_poly1305_finish (mbedtls_poly1305_context * ctx, unsigned char mac[16])

Parameters:

Parameter	Description
`ctx`	The Poly1305 context to use for the Poly1305 operation. This must be initialized and bound to a key.
`mac`	The buffer to where the MAC is written. This must be a writable buffer of length 16 bytes.

Returns:

0 on success.

A negative error code on failure.

3.18.7.2 void mbedtls_poly1305_free (mbedtls_poly1305_context * ctx)

Parameters:

Parameter	Description
`ctx`	The Poly1305 context to clear. This may be NULL , in which case this function is a no-op. If it is not NULL , it must point to an initialized Poly1305 context.

3.18.7.3 void mbedtls_poly1305_init (mbedtls_poly1305_context * ctx)

It must be the first API called before using the context.

It is usually followed by a call to **mbedtls_poly1305_starts()** , then one or more calls to **mbedtls_poly1305_update()** , then one call to **mbedtls_poly1305_finish()** , then finally **mbedtls_poly1305_free()**.

Parameters:

Parameter	Description
`ctx`	The Poly1305 context to initialize. This must not be NULL.

3.18.7.4 int mbedtls_poly1305_mac (const unsigned char key[32], const unsigned char * input, size_t ilen, unsigned char mac[16])

The key must be unique and unpredictable for each invocation of Poly1305.

Parameters:

Parameter	Description
`key`	The buffer containing the 32 byte (256 Bit) key.
`ilen`	The length of the input data in bytes. Any value is accepted.
`input`	The buffer holding the input data. This pointer can be NULL if ilen == 0.
`mac`	The buffer to where the MAC is written. This must be a writable buffer of length 16 bytes.

Returns:

0 on success.

A negative error code on failure.

3.18.7.5 int mbedtls_poly1305_starts (mbedtls_poly1305_context * ctx, const unsigned char key[32])

The key must be unique and unpredictable for each invocation of Poly1305.

Parameters:

Parameter	Description
`ctx`	The Poly1305 context to which the key should be bound. This must be initialized.
`key`	The buffer containing the 32 byte (256 Bit) key.

Returns:

0 on success.

A negative error code on failure.

3.18.7.6 int mbedtls_poly1305_update (mbedtls_poly1305_context * ctx, const unsigned char * input, size_t ilen)

It is called between mbedtls_cipher_poly1305_starts() and mbedtls_cipher_poly1305_finish(). It can be called repeatedly to process a stream of data.

Parameters:

Parameter	Description
`ctx`	The Poly1305 context to use for the Poly1305 operation. This must be initialized and bound to a key.
`ilen`	The length of the input data in bytes. Any value is accepted.
`input`	The buffer holding the input data. This pointer can be NULL if ilen == 0.

Returns:

0 on success.

A negative error code on failure.

3.19 rsa.h File reference

This file provides an API for the RSA public-key cryptosystem.

```
#include "config.h"

#include "bignum.h"

#include "md.h"
```

3.19.1 Data structures

- struct **mbedtls_rsa_context**

  The RSA context structure.

3.19.2 Macros

- #define **MBEDTLS_ERR_RSA_BAD_INPUT_DATA** -0x4080
- #define **MBEDTLS_ERR_RSA_INVALID_PADDING** -0x4100
- #define **MBEDTLS_ERR_RSA_KEY_GEN_FAILED** -0x4180
- #define **MBEDTLS_ERR_RSA_KEY_CHECK_FAILED** -0x4200
- #define **MBEDTLS_ERR_RSA_PUBLIC_FAILED** -0x4280
- #define **MBEDTLS_ERR_RSA_PRIVATE_FAILED** -0x4300
- #define **MBEDTLS_ERR_RSA_VERIFY_FAILED** -0x4380
- #define **MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE** -0x4400
- #define **MBEDTLS_ERR_RSA_RNG_FAILED** -0x4480
- #define **MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION** -0x4500
- #define **MBEDTLS_ERR_RSA_HW_ACCEL_FAILED** -0x4580
- #define **MBEDTLS_RSA_PUBLIC** 0
- #define **MBEDTLS_RSA_PRIVATE** 1
- #define **MBEDTLS_RSA_PKCS_V15** 0
- #define **MBEDTLS_RSA_PKCS_V21** 1
- #define **MBEDTLS_RSA_SIGN** 1
- #define **MBEDTLS_RSA_CRYPT** 2
- #define MBEDTLS_RSA_SALT_LEN_ANY -1

3.19.3 typedefs

- typedef struct **mbedtls_rsa_context mbedtls_rsa_context**

The RSA context structure.

3.19.4 Functions

- void **mbedtls_rsa_init** (**mbedtls_rsa_context** *ctx, int padding, int hash_id)

  This function initializes an RSA context.

- int **mbedtls_rsa_import** (**mbedtls_rsa_context** *ctx, const mbedtls_mpi *N, const mbedtls_mpi *P, const mbedtls_mpi *Q, const mbedtls_mpi *D, const mbedtls_mpi *E)

  This function imports a set of core parameters into an RSA context.

- int **mbedtls_rsa_import_raw** (**mbedtls_rsa_context** *ctx, unsigned char const *N, size_t N_len, unsigned char const *P, size_t P_len, unsigned char const *Q, size_t Q_len, unsigned char const *D, size_t D_len, unsigned char const *E, size_t E_len)

  This function imports core RSA parameters, in raw big-endian binary format, into an RSA context.

- int **mbedtls_rsa_complete** (**mbedtls_rsa_context** *ctx)

  This function completes an RSA context from a set of imported core parameters.

- int **mbedtls_rsa_export** (const **mbedtls_rsa_context** *ctx, mbedtls_mpi *N, mbedtls_mpi *P, mbedtls_mpi *Q, mbedtls_mpi *D, mbedtls_mpi *E)

  This function exports the core parameters of an RSA key.

- int **mbedtls_rsa_export_raw** (const **mbedtls_rsa_context** *ctx, unsigned char *N, size_t N_len, unsigned char *P, size_t P_len, unsigned char *Q, size_t Q_len, unsigned char *D, size_t D_len, unsigned char *E, size_t E_len)

  This function exports core parameters of an RSA key in raw big-endian binary format.

- int **mbedtls_rsa_export_crt** (const **mbedtls_rsa_context** *ctx, mbedtls_mpi *DP, mbedtls_mpi *DQ, mbedtls_mpi *QP)

  This function exports CRT parameters of a private RSA key.

- void **mbedtls_rsa_set_padding** (**mbedtls_rsa_context** *ctx, int padding, int hash_id)

  This function sets padding for an already initialized RSA context. See **mbedtls_rsa_init()** for details.

- size_t **mbedtls_rsa_get_len** (const **mbedtls_rsa_context** *ctx)

  This function retrieves the length of RSA modulus in bytes.

- int **mbedtls_rsa_gen_key** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, unsigned int nbits, int exponent)

  This function generates an RSA keypair.

- int **mbedtls_rsa_check_pubkey** (const **mbedtls_rsa_context** *ctx)

  This function checks if a context contains at least an RSA public key.

- int **mbedtls_rsa_check_privkey** (const **mbedtls_rsa_context** *ctx)

  This function checks if a context contains an RSA private key and perform basic consistency checks.

- int **mbedtls_rsa_check_pub_priv** (const **mbedtls_rsa_context** *pub, const **mbedtls_rsa_context** *prv)

  This function checks a public-private RSA key pair.

- int **mbedtls_rsa_public** (**mbedtls_rsa_context** *ctx, const unsigned char *input, unsigned char *output)

  This function performs an RSA public key operation.

- int **mbedtls_rsa_private** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, const unsigned char *input, unsigned char *output)

  This function performs an RSA private key operation.

- int **mbedtls_rsa_pkcs1_encrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  This function adds the message padding, then performs an RSA operation.

- int **mbedtls_rsa_rsaes_pkcs1_v15_encrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  This function performs a *PKCS#1 v1.5* encryption operation (RSAES-PKCS1-v1_5-ENCRYPT).

- int **mbedtls_rsa_rsaes_oaep_encrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t ilen, const unsigned char *input, unsigned char *output)

  This function performs a *PKCS#1 v2.1* OAEP encryption operation (RSAES-OAEP-ENCRYPT).

- int **mbedtls_rsa_pkcs1_decrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  This function performs an RSA operation, then removes the message padding.

- int **mbedtls_rsa_rsaes_pkcs1_v15_decrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  This function performs a *PKCS#1 v1.5* decryption operation (RSAES-PKCS1-v1_5-DECRYPT).

- int **mbedtls_rsa_rsaes_oaep_decrypt** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

This function performs a *PKCS#1 v2.1* OAEP decryption operation (RSAES-OAEP-DECRYPT).

- int **mbedtls_rsa_pkcs1_sign** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a private RSA operation to sign a message digest using PKCS#1.

- int **mbedtls_rsa_rsassa_pkcs1_v15_sign** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a *PKCS#1 v1.5* signature operation (RSASSA-PKCS1-v1_5-SIGN).

- int **mbedtls_rsa_rsassa_pss_sign** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a *PKCS#1 v2.1* PSS signature operation (RSASSA-PSS-SIGN).

- int **mbedtls_rsa_pkcs1_verify** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a public RSA operation and checks the message digest.

- int **mbedtls_rsa_rsassa_pkcs1_v15_verify** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a *PKCS#1 v1.5* verification operation (RSASSA-PKCS1-v1_5-VERIFY).

- int **mbedtls_rsa_rsassa_pss_verify** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a *PKCS#1 v2.1* PSS verification operation (RSASSA-PSS-VERIFY).

- int **mbedtls_rsa_rsassa_pss_verify_ext** (**mbedtls_rsa_context** *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, **mbedtls_md_type_t** md_alg, unsigned int hashlen, const unsigned char *hash, **mbedtls_md_type_t** mgf1_hash_id, int expected_salt_len, const unsigned char *sig)

  This function performs a *PKCS#1 v2.1* PSS verification operation (RSASSA-PSS-VERIFY).

- int **mbedtls_rsa_copy** (**mbedtls_rsa_context** *dst, const **mbedtls_rsa_context** *src)

  This function copies the components of an RSA context.

- void **mbedtls_rsa_free** (**mbedtls_rsa_context** *ctx)

  This function frees the components of an RSA key.

3.19.5 Detailed description

The RSA public-key cryptosystem is defined in *PKCS #1 v1.5 Public-Key Cryptography Standards RSA Encryption Standard* and *PKCS #1 v2.1 Public-Key Cryptography Standards RSA Cryptography Specifications*.

3.19.6 Macro definition documentation

3.19.6.1 #define MBEDTLS_ERR_RSA_BAD_INPUT_DATA -0x4080

Bad input parameters to function.

3.19.6.2 #define MBEDTLS_ERR_RSA_HW_ACCEL_FAILED -0x4580

RSA hardware accelerator failed.

3.19.6.3 #define MBEDTLS_ERR_RSA_INVALID_PADDING -0x4100

Input data contains invalid padding and is rejected.

3.19.6.4 #define MBEDTLS_ERR_RSA_KEY_CHECK_FAILED -0x4200

Key failed to pass the validity check of the library.

3.19.6.5 #define MBEDTLS_ERR_RSA_KEY_GEN_FAILED -0x4180

Something failed during generation of a key.

3.19.6.6 #define MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE -0x4400

The output buffer for decryption is not large enough.

3.19.6.7 #define MBEDTLS_ERR_RSA_PRIVATE_FAILED -0x4300

The private key operation failed.

3.19.6.8 #define MBEDTLS_ERR_RSA_PUBLIC_FAILED -0x4280

The public key operation failed.

3.19.6.9 #define MBEDTLS_ERR_RSA_RNG_FAILED -0x4480

The random generator failed to generate non-zeros.

3.19.6.10 #define MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION -0x4500

The implementation does not offer the requested operation, for example, because of security violations or lack of functionality.

3.19.6.11 #define MBEDTLS_ERR_RSA_VERIFY_FAILED -0x4380

The *PKCS#1* verification failed.

3.19.6.12 #define MBEDTLS_RSA_CRYPT 2

Identifier for RSA encryption and decryption operations.

3.19.6.13 #define MBEDTLS_RSA_PKCS_V15 0

Use *PKCS#1 v1.5* encoding.

3.19.6.14 #define MBEDTLS_RSA_PKCS_V21 1

Use *PKCS#1 v2.1* encoding.

3.19.6.15 #define MBEDTLS_RSA_PRIVATE 1

Request public key operation.

3.19.6.16 #define MBEDTLS_RSA_PUBLIC 0

Request private key operation.

3.19.6.17 #define MBEDTLS_RSA_SIGN 1

Identifier for RSA signature operations.

3.19.7 typedef documentation

3.19.7.1 typedef struct mbedtls_rsa_context mbedtls_rsa_context

Direct manipulation of the members of this structure is deprecated. All manipulation should instead be done through the public interface functions.

3.19.8 Function documentation

3.19.8.1 int mbedtls_rsa_check_privkey (const mbedtls_rsa_context * ctx)

The consistency checks performed by this function not only ensure that **mbedtls_rsa_private()** can be called successfully on the given context, but that the various parameters are mutually consistent with high probability, in the sense that **mbedtls_rsa_public()** and **mbedtls_rsa_private()** are inverses.

This function should catch accidental misconfigurations like swapping of parameters, but it cannot establish full trust in neither the quality nor the consistency of the key material that was used to setup the given RSA context: Consistency: Imported parameters that are irrelevant for the implementation might be silently dropped. If dropped, the current function does not have access to them, and therefore cannot check them. See **mbedtls_rsa_complete()**. If you want to check the consistency of the entire content of an *PKCS #1*-encoded RSA private key, for example, you should use mbedtls_rsa_validate_params() before setting up the RSA context. Additionally, if the implementation performs empirical checks, these checks substantiate but do not guarantee consistency. Quality: This function is not expected to perform extended quality assessments like checking that the prime factors are safe. Additionally, it is the responsibility of the user to ensure the trustworthiness of the source of his RSA parameters, which goes beyond what is effectively checkable by the library.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to check.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.2 int mbedtls_rsa_check_pub_priv (const mbedtls_rsa_context * pub, const mbedtls_rsa_context * prv)

It checks each of the contexts, and makes sure they match.

Parameters:

Parameter	Description
`pub`	The initialized RSA context holding the public key.
`prv`	The initialized RSA context holding the private key.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.3 int mbedtls_rsa_check_pubkey (const mbedtls_rsa_context * ctx)

If the function runs successfully, it is guaranteed that enough information is present to perform an RSA public key operation using **mbedtls_rsa_public()**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to check.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.4 int mbedtls_rsa_complete (mbedtls_rsa_context * ctx)

To setup an RSA public key, precisely N and E must have been imported.

To setup an RSA private key, sufficient information must be present for the other parameters to be derivable.

The default implementation supports the following:

- Derive P , Q from N , D , E.
- Derive N , D from P , Q , E.

Alternative implementations need not support these.

If this function runs successfully, it guarantees that the RSA context can be used for RSA operations without the risk of failure or crash.

This function need not perform consistency checks for the imported parameters. In particular, parameters that are not needed by the implementation might be silently discarded and left unchecked. To check the consistency of the key material, see **mbedtls_rsa_check_privkey()**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context holding imported parameters.

Returns:

0 on success.

MBEDTLS_ERR_RSA_BAD_INPUT_DATA if the attempted derivations failed.

3.19.8.5 int mbedtls_rsa_copy (mbedtls_rsa_context * dst, const mbedtls_rsa_context * src)

Parameters:

Parameter	Description
`dst`	The destination context. This must be initialized.
`src`	The source context. This must be initialized.

Returns:

0 on success.

#MBEDTLS_ERR_MPI_ALLOC_FAILED on memory allocation failure.

3.19.8.6 int mbedtls_rsa_export (const mbedtls_rsa_context * ctx, mbedtls_mpi * N, mbedtls_mpi * P, mbedtls_mpi * Q, mbedtls_mpi * D, mbedtls_mpi * E)

If this function runs successfully, the non-NULL buffers pointed to by N , P , Q , D , and E are fully written, with additional unused space filled leading by zero bytes.

Possible reasons for returning **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**:

- An alternative RSA implementation is in use, which stores the key externally, and either cannot or should not export it into RAM.
- A SW or HW implementation might not support a certain deduction. For example, P , Q from N , D , and E if the former are not part of the implementation.

If the function fails due to an unsupported operation, the RSA context stays intact and remains usable.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.
`N`	The MPI to hold the RSA modulus. This may be NULL if this field need not be exported.
`P`	The MPI to hold the first prime factor of N. This may be NULL if this field need not be exported.
`Q`	The MPI to hold the second prime factor of N. This may be NULL if this field need not be exported.
`D`	The MPI to hold the private exponent. This may be NULL if this field need not be exported.

Parameter	Description
E	The MPI to hold the public exponent. This may be NULL if this field need not be exported.

Returns:

0 on success.

MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED if exporting the requested parameters cannot be done due to missing functionality or because of security policies.

A non-zero return code on any other failure.

3.19.8.7 int mbedtls_rsa_export_crt (const mbedtls_rsa_context * ctx, mbedtls_mpi * DP, mbedtls_mpi * DQ, mbedtls_mpi * QP)

Alternative RSA implementations not using CRT-parameters internally can implement this function based on mbedtls_rsa_deduce_opt().

Parameters:

Parameter	Description
ctx	The initialized RSA context.
DP	The MPI to hold D modulo P-1 , or NULL if it need not be exported.
DQ	The MPI to hold D modulo Q-1 , or NULL if it need not be exported.
QP	The MPI to hold modular inverse of Q modulo P , or NULL if it need not be exported.

Returns:

0 on success.

A non-zero error code on failure.

3.19.8.8 int mbedtls_rsa_export_raw (const mbedtls_rsa_context * ctx, unsigned char * N, size_t N_len, unsigned char * P, size_t P_len, unsigned char * Q, size_t Q_len, unsigned char * D, size_t D_len, unsigned char * E, size_t E_len)

If this function runs successfully, the non-NULL buffers pointed to by N , P , Q , D , and E are fully written, with additional unused space filled leading by zero bytes.

Possible reasons for returning **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**:

- o An alternative RSA implementation is in use, which stores the key externally, and either cannot or should not export it into RAM.

- A SW or HW implementation might not support a certain deduction. For example, P , Q from N , D , and E if the former are not part of the implementation.

If the function fails due to an unsupported operation, the RSA context stays intact and remains usable.

The length parameters are ignored if the corresponding buffer pointers are NULL.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.
`N`	The byte array to store the RSA modulus, or NULL if this field need not be exported.
`N_len`	The size of the buffer for the modulus.
`P`	The byte array to hold the first prime factor of N , or NULL if this field need not be exported.
`P_len`	The size of the buffer for the first prime factor.
`Q`	The byte array to hold the second prime factor of N , or NULL if this field need not be exported.
`Q_len`	The size of the buffer for the second prime factor.
`D`	The byte array to hold the private exponent, or NULL if this field need not be exported.
`D_len`	The size of the buffer for the private exponent.
`E`	The byte array to hold the public exponent, or NULL if this field need not be exported.
`E_len`	The size of the buffer for the public exponent.

Returns:

0 on success.

MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED if exporting the requested parameters cannot be done due to missing functionality or because of security policies.

A non-zero return code on any other failure.

3.19.8.9 void mbedtls_rsa_free (mbedtls_rsa_context * ctx)

Parameters:

Parameter	Description
`ctx`	The RSA context to free. May be NULL , in which case this function is a no-op. If it is not NULL , it must point to an initialized RSA context.

3.19.8.10 int mbedtls_rsa_gen_key (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, unsigned int nbits, int exponent)

mbedtls_rsa_init() must be called before this function, to set up the RSA context.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context used to hold the key.
`f_rng`	The RNG function to be used for key generation. This must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context.
`nbits`	The size of the public key in bits.
`exponent`	The public exponent to use. For example, 65537. This must be odd and greater than 1.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.11 size_t mbedtls_rsa_get_len (const mbedtls_rsa_context * ctx)

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.

Returns:

The length of the RSA modulus in bytes.

3.19.8.12 int mbedtls_rsa_import (mbedtls_rsa_context * ctx, const mbedtls_mpi * N, const mbedtls_mpi * P, const mbedtls_mpi * Q, const mbedtls_mpi * D, const mbedtls_mpi * E)

This function can be called multiple times for successive imports, if the parameters are not simultaneously present.

Any sequence of calls to this function should be followed by a call to **mbedtls_rsa_complete()**, which checks and completes the provided information to a ready-for-use public or private RSA key.

See **mbedtls_rsa_complete()** for more information on which parameters are necessary to set up a private or public RSA key.

The imported parameters are copied and need not be preserved for the lifetime of the RSA context being set up.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to store the parameters in.
`N`	The RSA modulus. This may be NULL.
`P`	The first prime factor of N. This may be NULL.
`Q`	The second prime factor of N. This may be NULL.
`D`	The private exponent. This may be NULL.
`E`	The public exponent. This may be NULL.

Returns:

0 on success.

A non-zero error code on failure.

3.19.8.13 int mbedtls_rsa_import_raw (mbedtls_rsa_context * ctx, unsigned char const * N, size_t N_len, unsigned char const * P, size_t P_len, unsigned char const * Q, size_t Q_len, unsigned char const * D, size_t D_len, unsigned char const * E, size_t E_len)

This function can be called multiple times for successive imports, if the parameters are not simultaneously present. Any sequence of calls to this function should be followed by a call to **mbedtls_rsa_complete()**, which checks and completes the provided information to a ready-for-use public or private RSA key.

See **mbedtls_rsa_complete()** for more information on which parameters are necessary to set up a private or public RSA key.

The imported parameters are copied and need not be preserved for the lifetime of the RSA context being set up.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to store the parameters in.
`N`	The RSA modulus. This may be NULL.
`N_len`	The byte length of N ; it is ignored if N == NULL.
`P`	The first prime factor of N. This may be NULL.
`P_len`	The byte length of P ; it ns ignored if P == NULL.
`Q`	The second prime factor of N. This may be NULL.
`Q_len`	The byte length of Q ; it is ignored if Q == NULL.
`D`	The private exponent. This may be NULL.
`D_len`	The byte length of D ; it is ignored if D == NULL.
`E`	The public exponent. This may be NULL.
`E_len`	The byte length of E ; it is ignored if E == NULL.

Returns:

0 on success.

A non-zero error code on failure.

3.19.8.14 void mbedtls_rsa_init (mbedtls_rsa_context * ctx, int padding, int hash_id)

Set padding to **MBEDTLS_RSA_PKCS_V21** for the RSAES-OAEP encryption scheme and the RSASSA-PSS signature scheme.

The hash_id parameter is ignored when using **MBEDTLS_RSA_PKCS_V15** padding.

The choice of padding mode is strictly enforced for private key operations, since there might be security concerns in mixing padding modes. For public key operations it is a default value, which can be overriden by calling specific rsa_rsaes_xxx or rsa_rsassa_xxx functions.

The hash selected in hash_id is always used for OEAP encryption. For PSS signatures, it is always used for making signatures, but can be overriden for verifying them. If set to **MBEDTLS_MD_NONE**, it is always overriden.

Parameters:

Parameter	Description
`ctx`	The RSA context to initialize. This must not be NULL.
`padding`	The padding mode to use. This must be either **MBEDTLS_RSA_PKCS_V15** or **MBEDTLS_RSA_PKCS_V21**.
`hash_id`	The hash identifier of **mbedtls_md_type_t** type, if padding is **MBEDTLS_RSA_PKCS_V21**. It is unused otherwise.

3.19.8.15 int mbedtls_rsa_pkcs1_decrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

It is the generic wrapper for performing a *PKCS #1* decryption operation using the mode from the context.

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N (for example, 128 bytes if RSA-1024 is used) to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. If mode is **MBEDTLS_RSA_PUBLIC**, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).

Parameter	Description
`olen`	The address at which to store the length of the plaintext. This must not be NULL.
`input`	The ciphertext buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
`output`	The buffer used to hold the plaintext. This must be a writable buffer of length output_max_len bytes.
`output_max_len`	The length in bytes of the output buffer output.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.16 int mbedtls_rsa_pkcs1_encrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t ilen, const unsigned char * input, unsigned char * output)

It is the generic wrapper for performing a *PKCS #1* encryption operation using the mode from the context.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG to use. It is mandatory for *PKCS #1 v2.1* padding encoding, and for *PKCS #1 v1.5* padding encoding when used with mode set to **MBEDTLS_RSA_PUBLIC**. For *PKCS #1 v1.5* padding encoding and mode set to **MBEDTLS_RSA_PRIVATE**, it is used for blinding and should be provided in this case; see **mbedtls_rsa_private()** for more.
`p_rng`	The RNG context to be passed to f_rng. May be NULL if f_rng is NULL or if f_rng doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`ilen`	The length of the plaintext in bytes.

Parameter	Description
`input`	The input data to encrypt. This must be a readable buffer of size ilen bytes. This must not be NULL.
`output`	The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.17 int mbedtls_rsa_pkcs1_sign (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

It is the generic wrapper for performing a *PKCS #1* signature using the mode from the context.

The sig buffer must be as large as the size of ctx->N. For example, 128 bytes if RSA-1024 is used.

For *PKCS #1 v2.1* encoding, see comments on **mbedtls_rsa_rsassa_pss_sign()** for details on md_alg and hash_id.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function to use. If the padding mode is *PKCS #1 v2.1*, this must be provided. If the padding mode is *PKCS #1 v1.5* and mode is **MBEDTLS_RSA_PRIVATE**, it is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. It is ignored otherwise.

Parameter	Description
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. Ths is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer to hold the signature. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the signing operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.18 int mbedtls_rsa_pkcs1_verify (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

This is the generic wrapper for performing a *PKCS #1* verification using the mode from the context.

For *PKCS #1 v2.1* encoding, see comments on **mbedtls_rsa_rsassa_pss_verify()** about md_alg and hash_id.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA public key context to use.
`f_rng`	The RNG function to use. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. Otherwise, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. This is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer holding the signature. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the verify operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.19 int mbedtls_rsa_private (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, const unsigned char * input, unsigned char * output)

Blinding is used if and only if a PRNG is provided.

If blinding is used, both the base of exponentation and the exponent are blinded, providing protection against some side-channel attacks.

It is deprecated and a security risk to not provide a PRNG here and thereby prevent the use of blinding. Future versions of the library may enforce the presence of a PRNG.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function, used for blinding. It is discouraged and deprecated to pass NULL here, in which case blinding will be omitted.
`p_rng`	The RNG context to pass to f_rng. This may be NULL if f_rng is NULL or if f_rng doesn't need a context.
`input`	The input buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
`output`	The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.20 int mbedtls_rsa_public (mbedtls_rsa_context * ctx, const unsigned char * input, unsigned char * output)

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`input`	The input buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
`output`	The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

This function does not handle message padding.

Make sure to set input [0] = 0 or ensure that input is smaller than N.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.21 int mbedtls_rsa_rsaes_oaep_decrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, const unsigned char * label, size_t label_len, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N , for example, 128 bytes if RSA-1024 is used, to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns **MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE**.

Deprecated**:**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. If mode is **MBEDTLS_RSA_PUBLIC**, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).
`label`	The buffer holding the custom label to use. This must be a readable buffer of length label_len bytes. It may be NULL if label_len is 0.
`label_len`	The length of the label in bytes.
`olen`	The address at which to store the length of the plaintext. This must not be NULL.
`input`	The ciphertext buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
`output`	The buffer used to hold the plaintext. This must be a writable buffer of length output_max_len bytes.
`output_max_len`	The length in bytes of the output buffer output .

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.22 int mbedtls_rsa_rsaes_oaep_encrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, const unsigned char * label, size_t label_len, size_t ilen, const unsigned char * input, unsigned char * output)

The output buffer must be as large as the size of ctx->N. For example, 128 bytes if RSA-1024 is used.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initnialized RSA context to use.
`f_rng`	The RNG function to use. This is needed for padding generation and must be provided.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`label`	The buffer holding the custom label to use. This must be a readable buffer of length label_len bytes. It may be NULL if label_len is 0.
`label_len`	The length of the label in bytes.
`ilen`	The length of the plaintext buffer input in bytes.
`input`	The input data to encrypt. This must be a readable buffer of size ilen bytes. This must not be NULL.
`output`	The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.23 int mbedtls_rsa_rsaes_pkcs1_v15_decrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N , for example, 128 bytes if RSA-1024 is used, to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns **MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE**.

Deprecated**:**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. If mode is **MBEDTLS_RSA_PUBLIC**, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).
`olen`	The address at which to store the length of the plaintext. This must not be NULL.
`input`	The ciphertext buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
`output`	The buffer used to hold the plaintext. This must be a writable buffer of length output_max_len bytes.
`output_max_len`	The length in bytes of the output buffer output .

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.24 int mbedtls_rsa_rsaes_pkcs1_v15_encrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t ilen, const unsigned char * input, unsigned char * output)

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function to use. It is needed for padding generation if mode is **MBEDTLS_RSA_PUBLIC**. If mode is **MBEDTLS_RSA_PRIVATE** (discouraged), it is used for blinding and should be provided; see **mbedtls_rsa_private()**.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or if f_rng doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`ilen`	The length of the plaintext in bytes.
`input`	The input data to encrypt. This must be a readable buffer of size ilen bytes. This must not be NULL.
`output`	The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 on success.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.25 int mbedtls_rsa_rsassa_pkcs1_v15_sign (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. If mode is **MBEDTLS_RSA_PUBLIC**, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. This is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer to hold the signature. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the signing operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.26 int mbedtls_rsa_rsassa_pkcs1_v15_verify (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

Deprecated**:**

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA public key context to use.
`f_rng`	The RNG function to use. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. Otherwise, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. This is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer holding the signature. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the verify operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.27 int mbedtls_rsa_rsassa_pss_sign (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

The hash_id in the RSA context is the one used for the encoding. md_alg in the function call is the type of hash that is encoded. According to *PKCS #1 v2.1 Public-Key Cryptography Standards RSA Cryptography Specifications* it is advised to keep both hashes the same.

This function always uses the maximum possible salt size, up to the length of the payload hash. This choice of salt size complies with *FIPS 186-4 Digital Signature Standard (DSS)* §5.5 (e) and *PKCS #1 v2.2 Public-Key Cryptography Standards RSA Cryptography Standard* §9.1.1 step 3. Furthermore, this function enforces a minimum salt size which is the hash size minus 2 bytes. If this minimum size is too large given the key size (the salt size, plus the hash size, plus 2 bytes must be no more than the key size in bytes), this function returns **MBEDTLS_ERR_RSA_BAD_INPUT_DATA**.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PUBLIC** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PRIVATE**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PUBLIC** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to use.
`f_rng`	The RNG function. It must not be NULL.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng doesn't need a context argument.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PRIVATE** or **MBEDTLS_RSA_PUBLIC** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. Ths is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer to hold the signature. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the signing operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.28 int mbedtls_rsa_rsassa_pss_verify (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

The hash function for the MGF mask generating function is that specified in the RSA context.

The hash_id in the RSA context is the one used for the verification. md_alg in the function call is the type of hash that is verified. According to *PKCS #1 v2.1 Public-Key Cryptography Standards RSA Cryptography Specifications* it is advised to keep both hashes the same. If hash_id in the RSA context is unset, the md_alg from the function call is used.

Deprecated:

It is deprecated and discouraged to call this function in **MBEDTLS_RSA_PRIVATE** mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to **MBEDTLS_RSA_PUBLIC**.

Alternative implementations of RSA need not support mode being set to **MBEDTLS_RSA_PRIVATE** and might instead return **MBEDTLS_ERR_PLATFORM_FEATURE_UNSUPPORTED**.

Parameters:

Parameter	Description
`ctx`	The initialized RSA public key context to use.
`f_rng`	The RNG function to use. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. Otherwise, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE** (deprecated).
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. This is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`sig`	The buffer holding the signature. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the verify operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.29 int mbedtls_rsa_rsassa_pss_verify_ext (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, mbedtls_md_type_t mgf1_hash_id, int expected_salt_len, const unsigned char * sig)

The hash function for the MGF mask generating function is that specified in mgf1_hash_id .

The sig buffer must be as large as the size of ctx->N. For example, 128 bytes if RSA-1024 is used.

The hash_id in the RSA context is ignored.

Parameters:

Parameter	Description
`ctx`	The initialized RSA public key context to use.
`f_rng`	The RNG function to use. If mode is **MBEDTLS_RSA_PRIVATE**, this is used for blinding and should be provided; see **mbedtls_rsa_private()** for more. Otherwise, it is ignored.
`p_rng`	The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or doesn't need a context.
`mode`	The mode of operation. This must be either **MBEDTLS_RSA_PUBLIC** or **MBEDTLS_RSA_PRIVATE**.
`md_alg`	The message-digest algorithm used to hash the original data. Use **MBEDTLS_MD_NONE** for signing raw data.
`hashlen`	The length of the message digest. This is only used if md_alg is **MBEDTLS_MD_NONE**.
`hash`	The buffer holding the message digest or raw data. If md_alg is **MBEDTLS_MD_NONE**, this must be a readable buffer of length hashlen bytes. If md_alg is not **MBEDTLS_MD_NONE**, it must be a readable buffer of length the size of the hash corresponding to md_alg.
`mgf1_hash_id`	The message digest used for mask generation.
`expected_salt_len`	The length of the salt used in padding. Use #MBEDTLS_RSA_SALT_LEN_ANY to accept any salt length.
`sig`	The buffer holding the signature. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.

Returns:

0 if the verify operation was successful.

An MBEDTLS_ERR_RSA_XXX error code on failure.

3.19.8.30 void mbedtls_rsa_set_padding (mbedtls_rsa_context * ctx, int padding, int hash_id)

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to be configured.
`padding`	The padding mode to use. This must be either **MBEDTLS_RSA_PKCS_V15** or **MBEDTLS_RSA_PKCS_V21**.
`hash_id`	The **MBEDTLS_RSA_PKCS_V21** hash identifier.

3.20 sha1.h File reference

This file contains SHA-1 definitions and functions.

```
#include "config.h"

#include <stddef.h>

#include <stdint.h>
```

3.20.1 Data structures

- struct **mbedtls_sha1_context**

  The SHA-1 context structure.

3.20.2 Macros

- #define **MBEDTLS_ERR_SHA1_HW_ACCEL_FAILED** -0x0035
- #define **MBEDTLS_ERR_SHA1_BAD_INPUT_DATA** -0x0073
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

3.20.3 typedefs

- typedef struct **mbedtls_sha1_context mbedtls_sha1_context**

  The SHA-1 context structure.

3.20.4 Functions

- void **mbedtls_sha1_init** (**mbedtls_sha1_context** *ctx)

  This function initializes a SHA-1 context.

- void **mbedtls_sha1_free** (**mbedtls_sha1_context** *ctx)

This function clears a SHA-1 context.

- void **mbedtls_sha1_clone** (**mbedtls_sha1_context** *dst, const **mbedtls_sha1_context** *src)

  This function clones the state of a SHA-1 context.

- int **mbedtls_sha1_starts_ret** (**mbedtls_sha1_context** *ctx)

  This function starts a SHA-1 checksum calculation.

- int **mbedtls_sha1_update_ret** (**mbedtls_sha1_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-1 checksum calculation.

- int **mbedtls_sha1_finish_ret** (**mbedtls_sha1_context** *ctx, unsigned char output[20])

  This function finishes the SHA-1 operation, and writes the result to the output buffer.

- int **mbedtls_internal_sha1_process** (**mbedtls_sha1_context** *ctx, const unsigned char data[64])

  SHA-1 process data block (internal use only).

- MBEDTLS_DEPRECATED void **mbedtls_sha1_starts** (**mbedtls_sha1_context** *ctx)

  This function starts a SHA-1 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha1_update** (**mbedtls_sha1_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-1 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha1_finish** (**mbedtls_sha1_context** *ctx, unsigned char output[20])

  This function finishes the SHA-1 operation, and writes the result to the output buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha1_process** (**mbedtls_sha1_context** *ctx, const unsigned char data[64])

  SHA-1 process data block (internal use only).

- int **mbedtls_sha1_ret** (const unsigned char *input, size_t ilen, unsigned char output[20])

  This function calculates the SHA-1 checksum of a buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha1** (const unsigned char *input, size_t ilen, unsigned char output[20])

  This function calculates the SHA-1 checksum of a buffer.

3.20.5 Detailed description

The Secure Hash Algorithm 1 (SHA-1) cryptographic hash function is defined in *FIPS 180-4 Secure Hash Standard (SHS)*.

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

3.20.6 Macro definition documentation

3.20.6.1 #define MBEDTLS_ERR_SHA1_BAD_INPUT_DATA -0x0073

SHA-1 input data was malformed.

3.20.6.2 #define MBEDTLS_ERR_SHA1_HW_ACCEL_FAILED -0x0035

SHA-1 hardware accelerator failed

3.20.7 typedef documentation

3.20.7.1 typedef struct mbedtls_sha1_context mbedtls_sha1_context

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

3.20.8 Function documentation

3.20.8.1 int mbedtls_internal_sha1_process (mbedtls_sha1_context * ctx, const unsigned char data[64])

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to use. This must be initialized.
`data`	The data block being processed. This must be a readable buffer of length 64 bytes.

Returns:

0 on success.

A negative error code on failure.

3.20.8.2 MBEDTLS_DEPRECATED void mbedtls_sha1 (const unsigned char * input, size_t ilen, unsigned char output[20])

The function allocates the context, performs the calculation, and frees the context.

The SHA-1 result is calculated as output = SHA-1(input buffer).

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Deprecated:

Superseded by **mbedtls_sha1_ret()** in 2.7.0

Parameters:

Parameter	Description
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data input in bytes.
`output`	The SHA-1 checksum result. This must be a writable buffer of size 20 bytes.

3.20.8.3 void mbedtls_sha1_clone (mbedtls_sha1_context * dst, const mbedtls_sha1_context * src)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`dst`	The SHA-1 context to clone to. This must be initialized.
`src`	The SHA-1 context to clone from. This must be initialized.

3.20.8.4 MBEDTLS_DEPRECATED void mbedtls_sha1_finish (mbedtls_sha1_context * ctx, unsigned char output[20])

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Deprecated:

Superseded by **mbedtls_sha1_finish_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context. This must be initialized and have a hash operation started.
`output`	The SHA-1 checksum result. This must be a writable buffer of length 20 bytes.

3.20.8.5 int mbedtls_sha1_finish_ret (mbedtls_sha1_context * ctx, unsigned char output[20])

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to use. This must be initialized and have a hash operation started.
`output`	The SHA-1 checksum result. This must be a writable buffer of length 20 bytes.

Returns:

0 on success.

A negative error code on failure.

3.20.8.6 void mbedtls_sha1_free (mbedtls_sha1_context * ctx)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to clear. This may be NULL , in which case this function does nothing. If it is not NULL , it must point to an initialized SHA-1 context.

3.20.8.7 void mbedtls_sha1_init (mbedtls_sha1_context * ctx)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to initialize. This must not be NULL.

3.20.8.8 MBEDTLS_DEPRECATED void mbedtls_sha1_process (mbedtls_sha1_context * ctx, const unsigned char data[64])

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Deprecated:

Superseded by **mbedtls_internal_sha1_process()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context. This must be initialized.
`data`	The data block being processed. This must be a readable buffer of length 64 bytes.

3.20.8.9 int mbedtls_sha1_ret (const unsigned char * input, size_t ilen, unsigned char output[20])

The function allocates the context, performs the calculation, and frees the context.

The SHA-1 result is calculated as output = SHA-1(input buffer).

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data input in bytes.
`output`	The SHA-1 checksum result. This must be a writable buffer of length 20 bytes.

Returns:

0 on success.

A negative error code on failure.

3.20.8.10 MBEDTLS_DEPRECATED void mbedtls_sha1_starts (mbedtls_sha1_context * ctx)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Deprecated:

Superseded by **mbedtls_sha1_starts_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to initialize. This must be initialized.

3.20.8.11 int mbedtls_sha1_starts_ret (mbedtls_sha1_context * ctx)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to initialize. This must be initialized.

Returns:

0 on success.

A negative error code on failure.

3.20.8.12 MBEDTLS_DEPRECATED void mbedtls_sha1_update (mbedtls_sha1_context * ctx, const unsigned char * input, size_t ilen)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Deprecated:

Superseded by **mbedtls_sha1_update_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context. This must be initialized and have a hash operation started.
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data input in bytes.

3.20.8.13 int mbedtls_sha1_update_ret (mbedtls_sha1_context * ctx, const unsigned char * input, size_t ilen)

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context. This must be initialized and have a hash operation started.
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data input in bytes.

Returns:

0 on success.

A negative error code on failure.

3.21 sha256.h File reference

This file contains SHA-224 and SHA-256 definitions and functions.

```
#include "config.h"

#include <stddef.h>

#include <stdint.h>
```

3.21.1 Data structures

- struct **mbedtls_sha256_context**

  The SHA-256 context structure.

3.21.2 Macros

- #define **MBEDTLS_ERR_SHA256_HW_ACCEL_FAILED** -0x0037
- #define **MBEDTLS_ERR_SHA256_BAD_INPUT_DATA** -0x0074
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

3.21.3 typedefs

- typedef struct **mbedtls_sha256_context mbedtls_sha256_context**

  The SHA-256 context structure.

3.21.4 Functions

- void **mbedtls_sha256_init** (**mbedtls_sha256_context** *ctx)

  This function initializes a SHA-256 context.

- void **mbedtls_sha256_free** (**mbedtls_sha256_context** *ctx)

  This function clears a SHA-256 context.

- void **mbedtls_sha256_clone** (**mbedtls_sha256_context** *dst, const **mbedtls_sha256_context** *src)

  This function clones the state of a SHA-256 context.

- int **mbedtls_sha256_starts_ret** (**mbedtls_sha256_context** *ctx, int is224)

  This function starts a SHA-224 or SHA-256 checksum calculation.

- int **mbedtls_sha256_update_ret** (**mbedtls_sha256_context** *ctx, const unsigned char *input, size_t ilen)

This function feeds an input buffer into an ongoing SHA-256 checksum calculation.

- int **mbedtls_sha256_finish_ret** (**mbedtls_sha256_context** *ctx, unsigned char output[32])

  This function finishes the SHA-256 operation, and writes the result to the output buffer.

- int **mbedtls_internal_sha256_process** (**mbedtls_sha256_context** *ctx, const unsigned char data[64])

  This function processes a single data block within the ongoing SHA-256 computation. This function is for internal use only.

- MBEDTLS_DEPRECATED void **mbedtls_sha256_starts** (**mbedtls_sha256_context** *ctx, int is224)

  This function starts a SHA-224 or SHA-256 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha256_update** (**mbedtls_sha256_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-256 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha256_finish** (**mbedtls_sha256_context** *ctx, unsigned char output[32])

  This function finishes the SHA-256 operation, and writes the result to the output buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha256_process** (**mbedtls_sha256_context** *ctx, const unsigned char data[64])

  This function processes a single data block within the ongoing SHA-256 computation. This function is for internal use only.

- int **mbedtls_sha256_ret** (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-224 or SHA-256 checksum of a buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha256** (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-224 or SHA-256 checksum of a buffer.

3.21.5 Detailed description

The Secure Hash Algorithms 224 and 256 (SHA-224 and SHA-256) cryptographic hash functions are defined in *FIPS 180-4 Secure Hash Standard (SHS)*.

3.21.6 Macro definition documentation

3.21.6.1 #define MBEDTLS_ERR_SHA256_BAD_INPUT_DATA -0x0074

SHA-256 input data was malformed.

3.21.6.2 #define MBEDTLS_ERR_SHA256_HW_ACCEL_FAILED -0x0037

SHA-256 hardware accelerator failed

3.21.7 typedef documentation

3.21.7.1 typedef struct mbedtls_sha256_context mbedtls_sha256_context

The structure is used both for SHA-256 and for SHA-224 checksum calculations. The choice between these two is made in the call to **mbedtls_sha256_starts_ret()**.

3.21.8 Function documentation

3.21.8.1 int mbedtls_internal_sha256_process (mbedtls_sha256_context * ctx, const unsigned char data[64])

Parameters:

Parameter	Description
`ctx`	The SHA-256 context. This must be initialized.
`data`	The buffer holding one block of data. This must be a readable buffer of length 64 bytes.

Returns:

0 on success.

A negative error code on failure.

3.21.8.2 MBEDTLS_DEPRECATED void mbedtls_sha256 (const unsigned char * input, size_t ilen, unsigned char output[32], int is224)

The function allocates the context, performs the calculation, and frees the context.

The SHA-256 result is calculated as output = SHA-256(input buffer).

Deprecated:

Superseded by **mbedtls_sha256_ret()** in 2.7.0.

Parameters:

Parameter	Description
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.
`output`	The SHA-224 or SHA-256 checksum result. This must be a writable buffer of length 32 bytes.

Parameter	Description
`is224`	Determines which function to use. This must be either 0 for SHA-256, or 1 for SHA-224.

3.21.8.3 void mbedtls_sha256_clone (mbedtls_sha256_context * dst, const mbedtls_sha256_context * src)

Parameters:

Parameter	Description
`dst`	The destination context. This must be initialized.
`src`	The context to clone. This must be initialized.

3.21.8.4 MBEDTLS_DEPRECATED void mbedtls_sha256_finish (mbedtls_sha256_context * ctx, unsigned char output[32])

Deprecated:

Superseded by **mbedtls_sha256_finish_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context. This must be initialized and have a hash operation started.
`output`	The SHA-224 or SHA-256 checksum result. This must be a writable buffer of length 32 bytes.

3.21.8.5 int mbedtls_sha256_finish_ret (mbedtls_sha256_context * ctx, unsigned char output[32])

Parameters:

Parameter	Description
`ctx`	The SHA-256 context. This must be initialized and have a hash operation started.
`output`	The SHA-224 or SHA-256 checksum result. This must be a writable buffer of length 32 bytes.

Returns:

0 on success.

A negative error code on failure.

3.21.8.6 void mbedtls_sha256_free (mbedtls_sha256_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-256 context to clear. This may be NULL , in which case this function returns immediately. If it is not NULL , it must point to an initialized SHA-256 context.

3.21.8.7 void mbedtls_sha256_init (mbedtls_sha256_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-256 context to initialize. This must not be NULL.

3.21.8.8 MBEDTLS_DEPRECATED void mbedtls_sha256_process (mbedtls_sha256_context * ctx, const unsigned char data[64])

Deprecated**:**

Superseded by **mbedtls_internal_sha256_process()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context. This must be initialized.
`data`	The buffer holding one block of data. This must be a readable buffer of size 64 bytes.

3.21.8.9 int mbedtls_sha256_ret (const unsigned char * input, size_t ilen, unsigned char output[32], int is224)

The function allocates the context, performs the calculation, and frees the context.

The SHA-256 result is calculated as output = SHA-256(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.
`output`	The SHA-224 or SHA-256 checksum result. This must be a writable buffer of length 32 bytes.
`is224`	Determines which function to use. This must be either 0 for SHA-256, or 1 for SHA-224.

3.21.8.10 MBEDTLS_DEPRECATED void mbedtls_sha256_starts (mbedtls_sha256_context * ctx, int is224)

Deprecated:

Superseded by **mbedtls_sha256_starts_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The context to use. This must be initialized.
`is224`	Determines which function to use. This must be either 0 for SHA-256, or 1 for SHA-224.

3.21.8.11 int mbedtls_sha256_starts_ret (mbedtls_sha256_context * ctx, int is224)

Parameters:

Parameter	Description
`ctx`	The context to use. This must be initialized.
`is224`	This determines which function to use. This must be either 0 for SHA-256, or 1 for SHA-224.

Returns:

0 on success.

A negative error code on failure.

3.21.8.12 MBEDTLS_DEPRECATED void mbedtls_sha256_update (mbedtls_sha256_context * ctx, const unsigned char * input, size_t ilen)

Deprecated:

Superseded by **mbedtls_sha256_update_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context to use. This must be initialized and have a hash operation started.
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.

3.21.8.13 int mbedtls_sha256_update_ret (mbedtls_sha256_context * ctx, const unsigned char * input, size_t ilen)

Parameters:

Parameter	Description
`ctx`	The SHA-256 context. This must be initialized and have a hash operation started.
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.

Returns:

0 on success.

A negative error code on failure.

3.22 sha512.h File reference

This file contains SHA-384 and SHA-512 definitions and functions.

```
#include "config.h"

#include <stddef.h>

#include <stdint.h>
```

3.22.1 Data structures

- struct **mbedtls_sha512_context**

  The SHA-512 context structure.

3.22.2 Macros

- #define **MBEDTLS_ERR_SHA512_HW_ACCEL_FAILED** -0x0039
- #define **MBEDTLS_ERR_SHA512_BAD_INPUT_DATA** -0x0075
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

3.22.3 typedefs

- typedef struct **mbedtls_sha512_context mbedtls_sha512_context**

  The SHA-512 context structure.

3.22.4 Functions

- void **mbedtls_sha512_init** (**mbedtls_sha512_context** *ctx)

  This function initializes a SHA-512 context.
- void **mbedtls_sha512_free** (**mbedtls_sha512_context** *ctx)

  This function clears a SHA-512 context.
- void **mbedtls_sha512_clone** (**mbedtls_sha512_context** *dst, const **mbedtls_sha512_context** *src)

  This function clones the state of a SHA-512 context.
- int **mbedtls_sha512_starts_ret** (**mbedtls_sha512_context** *ctx, int is384)

  This function starts a SHA-384 or SHA-512 checksum calculation.
- int **mbedtls_sha512_update_ret** (**mbedtls_sha512_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512 checksum calculation.

- int **mbedtls_sha512_finish_ret** (**mbedtls_sha512_context** *ctx, unsigned char output[64])

  This function finishes the SHA-512 operation, and writes the result to the output buffer. This function is for internal use only.

- int **mbedtls_internal_sha512_process** (**mbedtls_sha512_context** *ctx, const unsigned char data[128])

  This function processes a single data block within the ongoing SHA-512 computation.

- MBEDTLS_DEPRECATED void **mbedtls_sha512_starts** (**mbedtls_sha512_context** *ctx, int is384)

  This function starts a SHA-384 or SHA-512 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha512_update** (**mbedtls_sha512_context** *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512 checksum calculation.

- MBEDTLS_DEPRECATED void **mbedtls_sha512_finish** (**mbedtls_sha512_context** *ctx, unsigned char output[64])

  This function finishes the SHA-512 operation, and writes the result to the output buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha512_process** (**mbedtls_sha512_context** *ctx, const unsigned char data[128])

  This function processes a single data block within the ongoing SHA-512 computation. This function is for internal use only.

- int **mbedtls_sha512_ret** (const unsigned char *input, size_t ilen, unsigned char output[64], int is384)

  This function calculates the SHA-512 or SHA-384 checksum of a buffer.

- MBEDTLS_DEPRECATED void **mbedtls_sha512** (const unsigned char *input, size_t ilen, unsigned char output[64], int is384)

  This function calculates the SHA-512 or SHA-384 checksum of a buffer.

3.22.5 Detailed description

The Secure Hash Algorithms 384 and 512 (SHA-384 and SHA-512) cryptographic hash functions are defined in *FIPS 180-4 Secure Hash Standard (SHS)*.

3.22.6 Macro definition documentation

3.22.6.1 #define MBEDTLS_ERR_SHA512_BAD_INPUT_DATA -0x0075

SHA-512 input data was malformed.

3.22.6.2 #define MBEDTLS_ERR_SHA512_HW_ACCEL_FAILED -0x0039

SHA-512 hardware accelerator failed

3.22.7 typedef documentation

3.22.7.1 typedef struct mbedtls_sha512_context mbedtls_sha512_context

The structure is used both for SHA-384 and for SHA-512 checksum calculations. The choice between these two is made in the call to **mbedtls_sha512_starts_ret()**.

3.22.8 Function documentation

3.22.8.1 int mbedtls_internal_sha512_process (mbedtls_sha512_context * ctx, const unsigned char data[128])

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized.
`data`	The buffer holding one block of data. This must be a readable buffer of length 128 bytes.

Returns:

0 on success.

A negative error code on failure.

3.22.8.2 MBEDTLS_DEPRECATED void mbedtls_sha512 (const unsigned char * input, size_t ilen, unsigned char output[64], int is384)

The function allocates the context, performs the calculation, and frees the context.

The SHA-512 result is calculated as output = SHA-512(input buffer).

Deprecated:

Superseded by **mbedtls_sha512_ret()** in 2.7.0

Parameters:

Parameter	Description
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.
`output`	The SHA-384 or SHA-512 checksum result. This must be a writable buffer of length 64 bytes.

Parameter	Description
`is384`	Determines which function to use. This must be either 0 for SHA-512, or 1 for SHA-384.

3.22.8.3 void mbedtls_sha512_clone (mbedtls_sha512_context * dst, const mbedtls_sha512_context * src)

Parameters:

Parameter	Description
`dst`	The destination context. This must be initialized.
`src`	The context to clone. This must be initialized.

3.22.8.4 MBEDTLS_DEPRECATED void mbedtls_sha512_finish (mbedtls_sha512_context * ctx, unsigned char output[64])

Deprecated**:**

Superseded by **mbedtls_sha512_finish_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized and have a hash operation started.
`output`	The SHA-384 or SHA-512 checksum result. This must be a writable buffer of size 64 bytes.

3.22.8.5 int mbedtls_sha512_finish_ret (mbedtls_sha512_context * ctx, unsigned char output[64])

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized and have a hash operation started.
`output`	The SHA-384 or SHA-512 checksum result. This must be a writable buffer of length 64 bytes.

Returns:

0 on success.

A negative error code on failure.

3.22.8.6 void mbedtls_sha512_free (mbedtls_sha512_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to clear. This may be NULL , in which case this function does nothing. If it is not NULL , it must point to an initialized SHA-512 context.

3.22.8.7 void mbedtls_sha512_init (mbedtls_sha512_context * ctx)

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to initialize. This must not be NULL.

3.22.8.8 MBEDTLS_DEPRECATED void mbedtls_sha512_process (mbedtls_sha512_context * ctx, const unsigned char data[128])

Deprecated:

Superseded by **mbedtls_internal_sha512_process()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized.
`data`	The buffer holding one block of data. This must be a readable buffer of length 128 bytes.

3.22.8.9 int mbedtls_sha512_ret (const unsigned char * input, size_t ilen, unsigned char output[64], int is384)

The function allocates the context, performs the calculation, and frees the context.

The SHA-512 result is calculated as output = SHA-512(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.
`output`	The SHA-384 or SHA-512 checksum result. This must be a writable buffer of length 64 bytes.
`is384`	Determines which function to use. This must be either 0 for SHA-512, or 1 for SHA-384.

Returns:

0 on success.

A negative error code on failure.

3.22.8.10 MBEDTLS_DEPRECATED void mbedtls_sha512_starts (mbedtls_sha512_context * ctx, int is384)

Deprecated:

Superseded by **mbedtls_sha512_starts_ret()** in 2.7.0

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to use. This must be initialized.
`is384`	Determines which function to use. This must be either 0 for SHA-512 or 1 for SHA-384.

3.22.8.11 int mbedtls_sha512_starts_ret (mbedtls_sha512_context * ctx, int is384)

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to use. This must be initialized.
`is384`	Determines which function to use. This must be either for SHA-512, or 1 for SHA-384.

Returns:

0 on success.

A negative error code on failure.

3.22.8.12 MBEDTLS_DEPRECATED void mbedtls_sha512_update (mbedtls_sha512_context * ctx, const unsigned char * input, size_t ilen)

Deprecated:

Superseded by **mbedtls_sha512_update_ret()** in 2.7.0.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized and have a hash operation started.
`input`	The buffer holding the data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.

3.22.8.13 int mbedtls_sha512_update_ret (mbedtls_sha512_context * ctx, const unsigned char * input, size_t ilen)

Parameters:

Parameter	Description
`ctx`	The SHA-512 context. This must be initialized and have a hash operation started.
`input`	The buffer holding the input data. This must be a readable buffer of length ilen bytes.
`ilen`	The length of the input data in bytes.

Returns:

0 on success.

A negative error code on failure.

4 Runtime integration tests

4.1 Common integration tests

This section describes common CryptoCell-312 integration tests.

You must implement a subset of a function to serve as an abstraction layer between the integration test and the operating system of your choice.

4.1.1 Platform HAL integration tests

Platform HAL is responsible for initializing the board, which might include mapping of addresses and toggling modules at boot time.

4.1.1.1 Test_ProjInit

This function initializes platform, that is, maps CryptoCell-312 HW base address and environment HW base address in processMap.

```
uint32_t Test_ProjInit(void)
```

Returns:

- 0: success.
- 1: failure.

4.1.1.2 Test_ProjFree

This function unmaps CryptoCell-312 HW base address and environment HW base address in processMap.

```
void Test_ProjFree(void)
```

4.1.1.3 Test_ProjPerformPowerOnReset

This function performs PoR of CryptoCell-312, AO, and environment registers.

```
void Test_ProjPerformPowerOnReset(void)
```

4.1.1.4 Test_ProjCheckLcs

This function reads the LCS register and verifies that the LCS value is correct.

```
uint32_t Test_ProjCheckLcs(uint32_t nextLcs)
```

Returns:

- 0 on success
- `0x00FFFF02` (defined in `test_proj_common.h`) on failure.

Table 4-1 Test_PalMapAddr parameters

I/O	Parameter	Description
I	`nextLcs`	The address of the LCS register.

4.1.2 Address-mapping integration tests

The main purpose of these tests is to map the physical address of CryptoCell-312 registers to the virtual address of the OS.

4.1.2.1 Test_PalGetDMABaseAddr

This function returns the start (base) address of the DMA region.

```
unsigned long Test_PalGetDMABaseAddr(void);
```

Returns:

- The DMA base address.
    - When Armv8-M is supported, the Non-secure DMA base address.

4.1.2.2 Test_PalGetDMABaseAddr_s

This function returns the start (base) address of the Secure DMA region.

```
unsigned long Test_PalGetDMABaseAddr_s(void);
```

Returns:

- The Secure DMA base address.

4.1.2.3 Test_PalGetUnmanagedBaseAddr

This function returns the unmanaged base address.

```
unsigned long Test_PalGetUnmanagedBaseAddr (void);
```

Returns:

- The unmanaged base address.
    - When Armv8-M is supported, the Non-secure unmanaged base address.

4.1.2.4 Test_PalGetUnmanagedBaseAddr_s

This function returns the Secure unmanaged base address.

```
unsigned long Test_PalGetUnmanagedBaseAddr_s(void);
```

Returns:

- The Secure unmanaged base address.

4.1.2.5 Test_PalMapAddr

This function maps a physical address to a virtual address.

```
void *Test_PalMapAddr(void *physAddr, void *startingAddr, const char *filename,
size_t size, uint8_t protAndFlagsBitMask)
```

The mapping function returns the address of the memory-mapped CryptoCell registers when the following conditions are both true:

- There is no Memory Management Unit.
- The access to the physical memory is straightforward.

Returns:

- A valid virtual address on success, or NULL on failure.

Table 4-2 Test_PalMapAddr parameters

I/O	Parameter	Description
I	`physAddr`	The physical address.
I	`startingAddr`	The preferred static address for mapping.
I	`filename`	The filename, when using a file-based system. The `/dev` memory device path that enables access to memory.
I	`size`	The contents of a file mapping are initialized using size bytes starting at the `startingAddr` offset in the file.
I	`protAndFlagsBitMask`	Optional flags for permissions.

4.1.2.6 Test_PalUnmapAddr

This function unmaps the given virtual address.

```
void Test_PalUnmapAddr(void *virtAddr, size_t size)
```

Table 4-3 Test_PalUnmapAddr parameters

I/O	Parameter	Description
I	`virtAddr`	The virtual address to unmap.
I	`size`	The size of memory to unmap.

4.1.3 Memory integration tests

The integration test only uses DMA-able continuous memory.

4.1.3.1 Test_PalMemInit

This function initializes DMA memory management.

When Armv8-M is supported, it initializes the Non-secure DMA memory management.

```
uint32_t Test_PalMemInit(unsigned long newDMABaseAddr,
             unsigned long newUnmanagedBaseAddr,
             size_t DMAsize);
```

Returns:

- 0 on success.
- 1 on failure.

I/O	Parameter	Description
I	`newDMABaseAddr`	The new DMA start address.
I	`newUnmanagedBaseAddr`	The new unmanaged start address.
I	`DMAsize`	The size of the DMA region.

4.1.3.2 Test_PalMemInit_s

This function initializes the Secure DMA memory management.

```
uint32_t Test_PalMemInit_s(unsigned long newDMABaseAddr_s,
               unsigned long newUnmanagedBaseAddr_s,
               size_t SDMAsize);
```

Returns:

- 0 on success.
- 1 on failure.

I/O	Parameter	Description
I	`newDMABaseAddr_s`	New Secure DMA start address.
I	`newUnmanagedBaseAddr`	New Secure unmanaged start address.
I	`DMAsize`	Secure DMA region size.

4.1.3.3 Test_PalMemFin_s

This function sets the SECURE memory management driver to its initial state.

```
uint32_t Test_PalMemFin_s(void);
```

Returns:

- 0 on success.
- 1 on failure.

4.1.3.4 Test_PalMalloc

This function allocates a buffer in memory.

When Armv8-M is supported, this function is used only for Non-secure memory allocations.

```
void *Test_PalMalloc(size_t size);
```

Returns:

- A pointer to the allocated memory.

I/O	Parameter	Description
I	`size`	The requested buffer size in bytes.

4.1.3.5 Test_PalMalloc_s

This function allocates a buffer in the Secure memory region.

```
void *Test_PalMalloc_s(size_t size);
```

Returns:

- A pointer to the allocated Secure memory.

I/O	Parameter	Description
I	`size`	The requested buffer size in bytes.

4.1.3.6 Test_PalFree

This function frees allocated memory pointed by `pvAddress`.

When Armv8-M is supported, this function is used only for Non-secure memory blocks.

```
void Test_PalFree(void *pvAddress);
```

I/O	Parameter	Description
I	`pvAddress`	A pointer to the allocated memory.

4.1.3.7 Test_PalFree_s

This function frees Secure allocated memory pointed by `pvAddress`.

```
void Test_PalFree_s(void *pvAddress);
```

I/O	Parameter	Description
I	`pvAddress`	A pointer to the allocated memory.

4.1.3.8 Test_PalRealloc

This function reallocates the memory block pointed by `pvAddress`.

```
void *Test_PalRealloc (void *pvAddress, size_t newSize);
```

If the function fails to allocate the requested block of memory:

1. A null pointer is returned.
2. The memory block pointed by argument `pvAddress` is not deallocated.

When Armv8-M is supported, this function is used only for Non-secure memory blocks.

Returns:

- A pointer to the new allocated memory.
- NULL on failure.

I/O	Parameter	Description
I	`pvAddress`	A pointer to the allocated memory.
I	`newSize`	New size of the memory block.

4.1.3.9 Test_PalRealloc_s

This function changes the size of a Secure memory block pointed by `pvAddress`.

```
void *Test_PalRealloc_s (void *pvAddress, size_t newSize);
```

If the function fails to allocate the requested block of memory:

1. A null pointer is returned.
2. The memory block pointed by argument `pvAddress` is not deallocated.

When Armv8-M is supported, this function is used only for Non-secure memory blocks.

Returns:

- A pointer to the new allocated memory.
- NULL on failure.

I/O	Parameter	Description
I	`pvAddress`	A pointer to the allocated memory.
I	`newSize`	The new size of the memory block.

4.1.3.10 Test_PalDMAContigBufferAlloc

This function allocates a DMA-contiguous buffer and returns its address.

```
void *Test_PalDMAContigBufferAlloc(size_t size);
```

Returns:

- An address of the allocated buffer.

Table 4-4 Test_PalDMAContigBufferAlloc parameters

I/O	Parameter	Description
I	`buffSize`	The buffer size in Bytes.

4.1.3.11 Test_PalDMAContigBufferAlloc_s

This function allocates a DMA-contiguous buffer in a Secure memory region and returns its address.

```
void *Test_PalDMAContigBufferAlloc_s(size_t size);
```

Returns:

- The address of the Secure allocated buffer.

Table 4-5 Test_PalDMAContigBufferAlloc_s parameters

I/O	Parameter	Description
I	`buffSize`	The buffer size in bytes.

4.1.3.12 Test_PalDMAContigBufferFree

This function frees resources that `Test_PalDMAContigBufferAlloc()` has previously allocated.

```
void Test_PalDMAContigBufferFree(void *pvAddress)
```

Table 4-6 Test_PalDMAContigBufferFree parameters

I/O	Parameter	Description
I	`pvAddress`	The address of the allocated buffer to free.

4.1.3.13 Test_PalDMAContigBufferFree_s

This function frees resources that `Test_PalDMAContigBufferAlloc_s()` has previously allocated.

```
void Test_PalDMAContigBufferFree_s(void *pvAddress)
```

Table 4-7 Test_PalDMAContigBufferFree_s parameters

I/O	Parameter	Description
I	`pvAddress`	The address of the allocated buffer to free.

4.1.4 Thread integration tests

4.1.4.1 Test_PalThreadCreate

This function creates a thread.

```
ThreadHandle Test_PalThreadCreate(
          size_t stackSize,
          void *(*threadFunc)(void *),
          void *args,
          const char *threadName,
          uint8_t nameLen,
          uint8_t DmaAble);
```

To destroy the thread, you must call `Test_PalThreadDestroy()`.

Returns:

- The `threadFunc` address on success. NULL on failure.

Table 4-8 Test_PalThreadCreate parameters

I/O	Parameter	Description
I	`stackSize`	The stack size in bytes.
I	`threadFunc`	The thread function.
I	`args`	The input arguments for the thread function.
I	`threadName`	The name of the thread.

I/O	Parameter	Description
I	`nameLen`	The length of the thread.
I	`DmaAble`	Determines whether the stack is DMA-able: • True - DMA-able. • False - not DMA-able.

4.1.4.2 Test_PalThreadDestroy

This function destroys a thread.

```
uint32_t Test_PalThreadDestroy(ThreadHandle threadHandle);
```

Returns:

- 0 on success. 1 on failure.

Table 4-9 Test_PalThreadDestroy parameters

I/O	Parameter	Description
I	`threadHandle`	The thread structure.

4.1.4.3 Test_PalThreadJoin

This function waits for a thread to terminate.

```
uint32_t Test_PalThreadJoin(ThreadHandle threadHandle, void *threadRet)
```

If that thread has already terminated, it returns immediately. Returns:

- 0 on success. 1 on failure.

Table 4-10 Test_PalThreadJoin parameters

I/O	Parameter	Description
I	`threadHandle`	The thread structure. Not in use for FreeRTOS.
I	`threadRet`	The status of the target thread.

4.1.5 Time integration tests

Implements time-sensitive functions that are based on the underlying operating system.

4.1.5.1 Test_PalDelay

This function suspends execution of the calling thread for microsecond intervals.

```
void Test_PalDelay(const uint32_t msec)
```

Table 4-11 Test_PalDelay parameters

I/O	Parameter	Description
I	`msec`	The time to suspend execution, in microseconds.

4.2 Runtime register integration tests

The following tests check access to CryptoCell-312 runtime registers.

4.2.1 RUNIT_READ_REG

This function reads the register value from `offset`.

```
RUNIT_READ_REG(offset)
```

Returns:

- The value of the register.

Table 4-12 RUNIT_READ_REG parameters

I/O	Parameter	Description
I	`offset`	The offset from the beginning of the register file.

4.2.2 RUNIT_WRITE_REG

This function writes the value set in `val` to register at `wordOffset`.

```
RUNIT_WRITE_REG(wordOffset, val)
```

Table 4-13 RUNIT_WRITE_REG parameters

I/O	Parameter	Description
I	`wordOffset`	The offset of the register to overwrite.
O	`val`	The new value to write.

4.3 Runtime OTP integration tests

OTP implementation is partner-specific.

To run the integration test on an FPGA or simulation environment, you must have an implementation of the OTP module. The following functions must be adapted to your implementation.

4.3.1 RUNIT_WRITE_OTP

This function writes the value set in `val` to the OTP at `wordOffset`.

```
RUNIT_WRITE_OTP(wordOffset, val)
```

Table 4-14 RUNIT_WRITE_OTP parameters

I/O	Parameter	Description
I	`wordOffset`	The offset of the OTP to overwrite.
O	`val`	The new value to write.

4.3.2 RUNIT_READ_OTP

This function reads the OTP value from `offset`.

```
RUNIT_READ_OTP(offset)
```

Returns:

- The value of the register.

Table 4-15 RUNIT_READ_OTP parameters

I/O	Parameter	Description
I	`offset`	The offset from the beginning of the OTP file.

4.4 Runtime flash integration tests

The flash layer allows implementing flash-like behavior in systems and configurations that do not have physical flash modules.

4.4.1 runIt_flashInit

This function initiates the flash module.

```
RsvItError_t runIt_flashInit(size_t flashSize)
```

It must be called before other flash operations. This function initiates all that is required to imitate flash operations.

Returns:

- `RUNIT_ERROR_OK` on success.
- `RUNIT_ERROR_FAIL` on failure.

Table 4-16 runIt_flashInit parameters

I/O	Parameter	Description
I	`flashSize`	The size of the flash to initialize.

4.4.2 runIt_flashFinalize

This function closes a resource that is allocated for the Flash PAL module.

```
RsvItError_t runIt_flashFinalize(void)
```

It can be used for deallocation or a type of reset. Returns:

- `RUNIT_ERROR_OK` on success.

- `RUNIT_ERROR_FAIL` on failure.

4.4.3 runIt_flashWrite

This function writes to flash at the offset set in `addr`.

```
RunItError_t runIt_flashWrite(uint32_t addr, uint8_t* buff, size_t len)
```

Returns:

- `RUNIT_ERROR_OK` on success.
- `RUNIT_ERROR_FAIL` on failure.

Table 4-17 runIt_flashWrite parameters

I/O	Parameter	Description
I	`addr`	The offset from the start of the flash.
I	`buf`	The buffer to write to flash.
I	`len`	The length of data to write to flash.

4.4.4 runIt_flashRead

This function reads from flash at the `addr` address and writes to the `buf` buffer.

```
RunItError_t runIt_flashRead(uint32_t addr, uint8_t* buff, size_t len)
```

Returns:

- `RUNIT_ERROR_OK` on success.
- `RUNIT_ERROR_FAIL` on failure.

Table 4-18 runIt_flashRead parameters

I/O	Parameter	Description
I	`addr`	The offset from the start of the flash.
O	`buf`	The buffer to fill with read data.
I	`len`	The length of data to read from flash.

4.5 Runtime logging integration tests

Log entries are embedded in the integration test and are intended to debug and output the test result to your chosen output.

4.5.1 RUNIT_PRINT

This function prints a log entry.

```
RUNIT_PRINT(format, ...)
```

Table 4-19 RUNIT_PRINT parameters

I/O	Parameter	Description
I	`format`	The preferred output format.
I	`...`	Format arguments.

4.5.2 RUNIT_PRINT_HEADER

This function prints the header of the integration test.

```
RUNIT_PRINT_HEADER()
```

4.5.3 RUNIT_PRINT_DBG

This function prints a debug message.

```
RUNIT_PRINT_DBG(format, ...)
```

It is skipped unless compiled with `TEST_DEBUG`.

Table 4-20: RUNIT_PRINT_DBG parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

4.5.4 RUNIT_PRINT_ERROR

This function prints an error message to log.

```
RUNIT_PRINT_ERROR(format, ...)
```

Table 4-21 RUNIT_PRINT_ERROR parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

4.5.5 RUNIT_PRINT_BUF

This function prints the buffer of length `_size` with a `_label` label.

```
RUNIT_PRINT_BUF(_buf, _size, _label)
```

Table 4-22 RUNIT_PRINT_BUF parameters

I/O	Parameter	Description
I	`_buf`	The buffer to print.
I	`_size`	The size of the buffer to print.
I	`_label`	The label to show in the printout.

4.5.6 RUNIT_TEST_START

This function starts the test.

```
RUNIT_TEST_START(testName)
```

It is called at the beginning of every test. You can configure it to print a formatted line that indicates the test has started.

Table 4-23 RUNIT_TEST_START parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.5.7 RUNIT_TEST_RESULT

This function returns the test result.

```
RUNIT_TEST_RESULT(testName)
```

It is called at the end of every test. You can configure it to print a formatted line that indicates if the test has ended successfully.

Table 4-24 RUNIT_TEST_RESULT parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.5.8 RUNIT_SUB_TEST_START

This function starts the subtest.

```
RUNIT_SUB_TEST_START(_testName)
```

It is called at the beginning of every subtest. You can configure it to print a formatted line that indicates the subtest has started.

Table 4-25 RUNIT_SUB_TEST_START parameters

I/O	Parameter	Description
I	`_testName`	The name of the subtest.

4.5.9 RUNIT_SUB_TEST_RESULT

This function returns the subtest result.

```
RUNIT_SUB_TEST_RESULT(_testName)
```

It is called at the end of every subtest. You can configure it to print a formatted line that indicates whether the subtest has ended successfully.

Table 4-26 RUNIT_SUB_TEST_RESULT parameters

I/O	Parameter	Description
I	`_testName`	The name of the subtest.

4.5.10 RUNIT_SUB_TEST_RESULT_W_PARAMS

This function returns the subtest result, with optional test-specific information.

```
RUNIT_SUB_TEST_RESULT_W_PARAMS(_testName, _format, ...)
```

It is called at the end of every subtest. You can configure it to print a formatted line that indicates whether the subtest has ended successfully.

Table 4-27 RUNIT_SUB_TEST_RESULT_W_PARAMS parameters

I/O	Parameter	Description
I	`_testName`	The name of the subtest.
I	`_format`	The entry format.
I	`...`	Format arguments.

4.6 Production tool integration tests

4.6.1 ICV factory tools integration tests

The runtime integration tests package includes a reference application that tests the ICV factory tools.

For more details, see: `host/src/tests/integration_cc3x/cmpu_integration_test/cmpu_integration_test.c`.

4.6.1.1 CMPUIT_PRINT

This function prints a log entry.

```
CMPUIT_PRINT(format, ...)
```

Table 4-28 CMPUIT_PRINT parameters

I/O	Parameter	Description
I	`format`	The preferred output format.
I	`...`	Format arguments.

4.6.1.2 CMPUIT_TEST_START

This function starts the test.

```
CMPUIT_TEST_START(testName)
```

It is called at the beginning of every test. You can configure it to print a formatted line that indicates the test has started.

Table 4-29 CMPUIT_TEST_START parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.6.1.3 CMPUIT_TEST_RESULT

This function returns the test result.

```
CMPUIT_TEST_RESULT(testName)
```

It is called at the end of every test. You can configure it to print a formatted line that indicates if the test has ended successfully.

Table 4-30 CMPUIT_TEST_RESULT parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.6.1.4 CMPUIT_PRINT_ERROR

This function prints an error message to log.

```
CMPUIT_PRINT_ERROR(format, ...)
```

Table 4-31 CMPUIT_PRINT_ERROR parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

4.6.1.5 CMPUIT_PRINT_DBG

This function prints a debug message.

```
CMPUIT_PRINT_DBG(format, ...)
```

It is skipped unless compiled with `TEST_DEBUG`.

Table 4-32 CMPUIT_PRINT_DBG parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

4.6.2 OEM factory tools integration tests

The runtime integration tests package includes a reference application that tests the OEM factory tools.

For more details, see `host/src/tests/integration_cc3x/dmpu_integration_test/dmpu_integration_test.c`.

4.6.2.1 DMPUIT_PRINT

This function prints a log entry.

```
DMPUIT_PRINT(format, ...)
```

Table 4-33 DMPUIT_PRINT parameters

I/O	Parameter	Description
I	`format`	The preferred output format.
I	`...`	Format arguments.

4.6.2.2 DMPUIT_TEST_START

This function starts the test.

```
DMPUIT_TEST_START(testName)
```

It is called at the beginning of every test. You can configure it to print a formatted line that indicates the test has started.

Table 4-34 DMPUIT_TEST_START parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.6.2.3 DMPUIT_TEST_RESULT

This function returns the test result.

```
DMPUIT_TEST_RESULT(testName)
```

It is called at the end of every test. You can configure it to print a formatted line that indicates if the test has ended successfully.

Table 4-35 DMPUIT_TEST_RESULT parameters

I/O	Parameter	Description
I	`testName`	The name of the test.

4.6.2.4 DMPUIT_PRINT_ERROR

This function prints an error message to log.

```
DMPUIT_PRINT_ERROR(format, ...)
```

Table 4-36 DMPUIT_PRINT_ERROR parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

4.6.2.5 DMPUIT_PRINT_DBG

This function prints a debug message.

```
DMPUIT_PRINT_DBG(format, ...)
```

It is skipped unless compiled with `TEST_DEBUG`.

Table 4-37 DMPUIT_PRINT_DBG parameters

I/O	Parameter	Description
I	`format`	The entry format.
I	`...`	Format arguments.

Appendix A Revisions

Table A-1 Differences between 100777 Issue 0102-00 Runtime APIs and 101122 Issue 0103-01

Change	Location	Affects
Removed the `secureboot_basetypes.h` file.	**Runtime API layer**	r1p3
Some CryptoCell header files are removed and replaced by the Mbed TLS implementation, with CryptoCell H/W acceleration: • `mbedtls_cc_aes_crypt_additional.h` • `mbedtls_cc_aes_key_wrap.h` • `mbedtls_cc_aes_key_wrap_error.h` • `mbedtls_cc_chacha.h` • `mbedtls_cc_chacha_error.h` • `mbedtls_cc_chacha_poly.h` • `mbedtls_cc_chacha_poly_error.h` • `mbedtls_cc_hkdf.h` • `mbedtls_cc_hkdf_error.h` • `mbedtls_cc_poly.h` • `mbedtls_cc_poly_error.h`	**Runtime API layer**	r1p3
Some header files were reduced by the Mbed TLS implementation, with CryptoCell H/W acceleration: • `mbedtls_cc_aes_ccm_star.h`	**Runtime API layer**	r1p3
Removed `CC_ECPKI_DomainID_Builded` from the `CCEcpkiDomainID_t` enum in `cc_ecpki_types.h`.	Runtime API layer	r1p3
Some Mbed TLS header files are added: • `hkdf.h` • `nist_kw.h` • `chacha20.h` • `poly1305.h` • `chachapoly.h`	**Mbed TLS cryptographic API layer**	r1p3
Added the **Runtime integration tests** chapter (moved from Software Integrators Manual).	Entire document	r1p3

Change	Location	Affects
Added the following tests: • `Test_PalGetDMABaseAddr` • `Test_PalGetDMABaseAddr_s` • `Test_PalGetUnmanagedBaseAddr` • `Test_PalGetUnmanagedBaseAddr_s` • `Test_PalMemInit` • `Test_PalMemInit_s` • `Test_PalMemFin_s` • `Test_PalMalloc` • `Test_PalMalloc_s` • `Test_PalFree` • `Test_PalFree_s` • `Test_PalRealloc` • `Test_PalRealloc_s` • `Test_PalDMAContigBufferAlloc_s` • `Test_PalDMAContigBufferFree_s`	**Runtime integration tests**	r1p3