Introduction to RSS extension for Key Wrap (RSSe KW) service on STM32 MCUs

Introduction

This application note introduces the RSS extension Key Wrap (RSSe KW) feature available on the STM32 MCUs listed in Table 1.

In a context of symmetric or asymmetric cryptography, protection of secret keys is essential. For this purpose, ST proposes a solution so that the keys remain secret at any time and usable only on a specific device.

On STM32, two elements are required to do so. The first one is a hardware coupling and chaining bridge (CCB) to isolate the key manipulation inside hardware cryptographic IPs from any master on the system. The second one is a Root Security Services extension (RSSe) library allowing the user firmware to use the secret key while never giving access to the secret key itself. In the document, the library is referenced as “RSSe KW” and the output is referenced as “wrapped Key”.

The provisioning of unique keys may be required for some use cases as initial attestation, for example. This provisioning is a crucial step and may be very costly to guaranty the unicity and confidentiality of the keys. STMicroelectronics addresses this challenge by provisioning, in secure facilities, each applicable STM32 (cf. Table 1) unique key pairs called Device Unique Authentication keys (DUA). These keys are ECC (Elliptic Curve Cryptographic) keys of 256 bits. The public part is freely accessible through a certificate signed by an STMicroelectronics Certificate Authority (CA) while the wrapped private part is accessible only through the RSSe KW.

This application note gives an overview of the STM32 RSSe KW solution with its associated tools ecosystem and explains how to use it to safely manipulate secret keys for cryptographic operations.

Table 1. Applicable products

Type	Products
Microcontroller	STM32U385VG, STM32U385RGU, STM32U385RG, STM32U385RE, STM32U385RC, STM32U385KG, STM32U385CG

Note: In this document, the STM32U385xx devices are referred as STM32U3.

1 General information

This document applies to the STM32U3 series Arm® Cortex®-M33-based microcontrollers.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 References

Table 2. List of documents

Reference	Name	Title
[1]	AN3155	USART protocol used in the STM32 bootloader
[2]	AN3156	USB DFU protocol used in the STM32 bootloader
[3]	AN4221	I2C protocol used in the STM32 bootloader
[4]	AN4286	SPI protocol used in the STM32 bootloader
[5]	AN5927	I3C protocol used in the STM32 bootloader
[6]	RM0487	STM32U3 series Arm®-based 32-bit MCUs
[7]	UM2237	STM32CubeProgrammer software description
[8]	UM2238	STM32 Trusted Package Creator tool software description

1.2 Glossary

Acronym	Definition
AES	Advanced Encryption Standard
AES CBC	AES Cipher Block Chaining mode
AES ECB	AES Electronic Code Block mode
AES GCM	AES Gallois Counter Mode
CA	Certificate Authority
DUA	Device Unique Authentication
ECC	Elliptic Curve Cryptography
KW	Key Wrapping
MAC	Message Authentication Code
MCU	Microcontroller Unit
OB	Option Byte
RDP	Readout Protection
RSS	Root Security Services
RSSe KW	RSS extension Key Wrap
RSSe SFI	RSS extension Secure Firmware Install
Secure Boot	Root of Trust, checks STM32 security protection
Secure bootloader	Standard ST bootloader with additional security features
STM32 TPC	STM32 Trusted Package Creator (see document [8])
TZEN	TrustZone® enable option byte

2 STM32 RSSe Key Wrapping

2.1 RSSe KW principles overview

RSSe KW is a library implemented in STM32 microcontrollers that get wrapped keys in trusted facilities. The RSSe KW can be used through debugger access. The main goal of the RSSe KW is to provide wrapped keys. Here wrapped means usable only on one specific device.

The RSSe KW prevents the sensitive keys from:

- Being accessed by any firmware or debugging access
- Being used by another device (including STM32)

For the moment, the main goal of the RSSe KW is to provide wrapped DUA keys usable only on the current chip for a context given by the user. The DUA keys are provisioned in a trusted environment in ST facilities.

2.2 RSSe KW features

2.2.1 Internal key wrapped

The STM32U3 embeds two unique key pairs called the DUA. These pairs are provisioned at ST level in trusted facilities. The public part is accessible to the user through a certificate signed by a STMicroelectronics CA. The private part is never accessible by a user application nor a debugger access. The RSSe KW aims to provide wrapped DUA private keys. The private key is provided inside a container that contains all necessary informations to be used by the CCB in the customer's application. For more details about the container format, refer to Section 3.3.5: RSSE_KEYWRAP_ECC_CHIP_PRIVATE_KEY_ID.

2.3 RSSe KW constraints

The RSSe KW only runs on crypto parts.

One shot service: once installed, the RSSe KW allows only one service to be called (meaning one wrapped key structure at a time). To get another wrapped key, the user must uninstall the RSSe KW (by doing a reset on the board), then, critically, do a RDP regression (RDP = 0xAA) and reinstall it, thus run the procedure from the beginning (as detailed later).

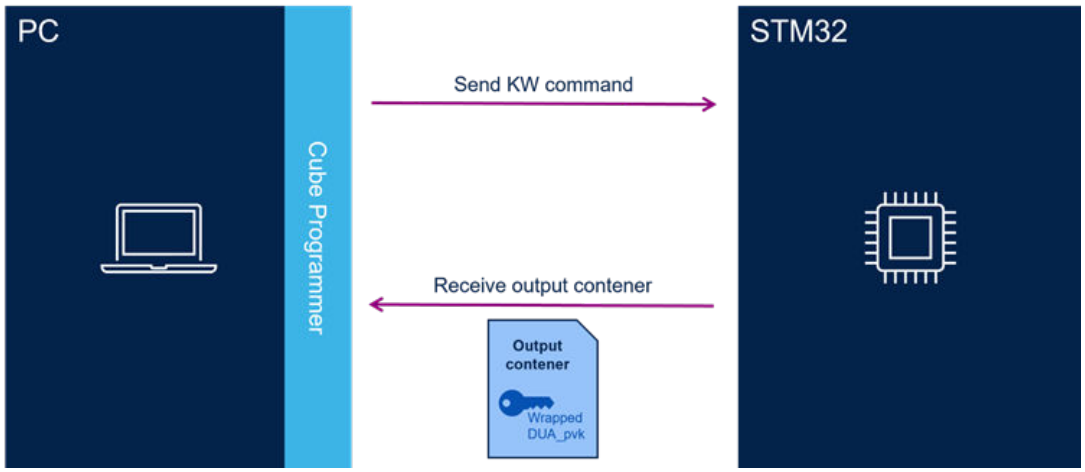
3 Interface

3.1 Cube Programmer interface

The Cube Programmer command to get wrapped DUA keys from the device is detailed in Cube Programmer documentation [7].

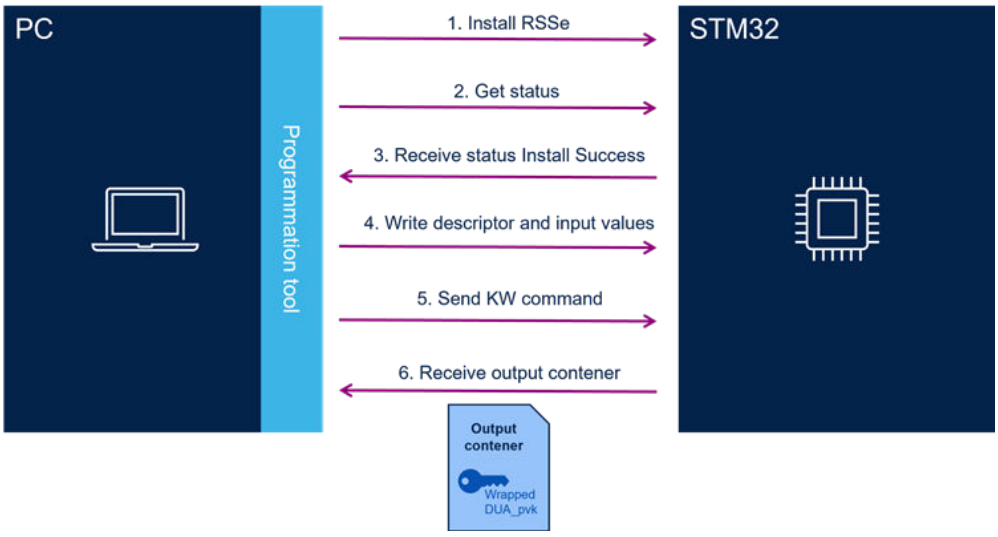
With the use of Cube Programmer, the user can get the wrapped key with only one command as shown in Figure 1. The output container is stored into a binary file chosen by the user.

Figure 1. Cube Programmer interface with user



With a custom tool, the user must follow the steps detailed in Figure 2 to get the wrapped key. This flow is also detailed in Section 3.3.5.

Figure 2. Detailed programming tool interface with user



3.2 Tool/ RSSLIB interface

In case Cube Programmer cannot be used to get the wrap key, the programing tool must follow the steps below.

3.2.1 Key wrap flow overview

The user needs to be equipped with a programming tool. To get a specific wrapped key, the tool must perform the following steps:

1. Install the RSSe KW.
   a. Set mandatory OB:
      i. TZEN = 1
      ii. RDP = 0x55
      iii. SRAM2_RST = 0
   b. Get the encrypted RSSe KW binary from the X-Cube RSSe package and download it into SRAM download area.
   c. Write the address of the encrypted RSSe KW binary to be installed at the address expected by the descriptor.
   d. Set the RSSCMD value in download area and the address expected by the descriptor “input address”.
   e. Set PC to RSSLIB_Set RSSCMD function call or send BL special command for RSSCMD setting.
2. Get RSSe KW installation status.
3. Write descriptor parameters for key wrapping.
   a. Input address to input structure
   b. Output address to expected output structure
4. Write input structure at the expected address.
5. Call RSSLIB_RSSeCall API with RSSE_KEYWRAP_ECC_CHIP_PRIVATE_KEY_ID.
6. Read the output structure at the expected address.

The steps are detailed in the following parts of this document.

3.2.2 STM32 descriptor

The STM32 system flash memory embeds a structure called STM32 descriptor. This structure provides host information regarding the STM32 device and the way the host can communicate with RSS.

The following table gives the C structure that describes the STM32 descriptor fields.

Table 3. stm32_descriptor_t description

Field name	Type	Description
Version	uint32_t	Descriptor version
CertificateAddr	uint32_t	Address of the chip certificate
AvailableRamStartAddr	uint32_t	Start address of the available SRAM for host
AvailableRamEndAddr	uint32_t	End address of the available SRAM for host
RsseLoadAddr	uint32_t	Address where to find address of the RSSe to install
RsseMaxSize	uint32_t	Maximum size of an RSSe
RsseParamAddr	uint32_t	Address where to find the input parameter address for the API call
RsseParamMaxSize	uint32_t	Maximum size of an RSSe parameter
RsseResultAddr	uint32_t	Address where to find RSSe call result value
Reserved	uint32_t[7]	Reserved
RSSLIBApiTable	RSSLIB_STM32xxApiTable_t	RSSlib API table within the RSSLIB system flash memory area

The address of STM32 descriptor field is provided within STM32 system memory at a specific address tied to the product.

Table 4. **STM32 descriptor address per product**

Product lines	STM32 descriptor address
STM32U385	0x0BF99000

3.2.3 RSS library API description

The root secure services library provides services for both secure and nonsecure firmware.

These secure services are split between secure firmware that can only call secure services and the nonsecure firmware that can only call nonsecure service.

The address of secure services is provided within a specific area named RSSLIB_STM32xxApiTable within STM32 system memory at a specific address tied to the product.RSSLIB_STM32xxApiTable address per product.

Table 5. **RSSLIB_STM32xxApiTable address per product**

Product lines	RSSLIB_STM32xxApiTable address
STM32U385	0x0BF99040

The following table gives the C structure that describes the RSSLIB_STM32xxApiTable dedicated to the supported STM32 microcontrollers.

Table 6. **RSSLIB_STM32ApiTable_t structure description**

Field name(1)	Type	Callable Attribute(2)	Description
RSSLIB_RSSeCall	RSSLIB_RSSeCall_t	NS	Gateway to call RSSe API
RSSLIB_Call	RSSLIB_Call_t	NS	Gateway to call RSSLIB API
Reserved	uint32_t[8]	NA	NA
RSSLIB_sec_CloseExitHDP	RSSLIB_sec_CloseExitHDP_t	S	Close and exit from flash memory HDP area jumping on passed vector table address (More details in product reference manual.)
RSSLIB_sec_CloseExitHDPExt	RSSLIB_sec_CloseExitHDPExt_t	S	Close and exit from flash memory HDPExt area jumping on passed vector table address (More details in product reference manual.)

1. *Each field of RSSLIB_STM32xxApiTable_t is a pointer to a function.*
2. *NS: nonsecure callable only; S: secure callable only*

The RSSLIB_RSSeCall & RSSLIB_Call fields listed in the above table are function types described in the following section.

By default, nonsecure callable services are not defined within a nonsecure callable SAU region. Hence, the user has no SAU parameters nor region to set in order to deny such secure services to nonsecure firmware.

3.2.4 RSSLIB API function types

This section provides the function types in C for RSSLIB API not described within product reference manual and that are relevant for SFI.

3.2.4.1 RSSLIB_Call

Service description: call a RSSlib service using the command identifier (CmdID) of the RSSlib service

Input parameter: payload_addr, type RSSLIB_RSSCmd_t

Output parameter: parameter stored in the address pointed by `STM32_descriptor→RsseResultAddr`. For example, if `STM32_descriptor->RsseResultAddr = 0x3000800C`, the RSSlib stores the API result at the address pointed by 0`0x3000800C`. The output parameter type depends on the RSSlib service called (described in RSSLIB_SetRSSCMD).

Return value: `RSSLIB_RSS_SUCCESS`, `RSSLIB_RSS_ERROR`

function type: `typedef uint32_t (*RSSLIB_Call_t)(RSSLIB_RSSCmd_t *payload_addr);`

Table 7. RSSLIB_RSSCmd_t description

Field name	Type	Comment
CmdID	uint32_t	Command ID
InputParamAddr	uint32_t	API input parameter address

3.2.4.2 *RSSLIB_RSSeCall*

Service description: call a RSSe service using the command identifier (CmdID) of the RSSe service

Input parameter: payload_addr, type RSSLIB_RSSeCmd_t

Output parameter: parameter stored in the address pointed by `STM32_descriptor→RsseResultAddr`. For example, if `STM32_descriptor->RsseResultAddr = 0x3000800C`, the RSSe stores the API result at the address pointed by 0`0x3000800C`. The output parameter type depends on the RSSe service called (described in RSSe_KEYWRAP_GetVersion).

Return value: `RSSLIB_RSS_SUCCESS`, `RSSLIB_RSS_ERROR`

function type: `typedef uint32_t (*RSSLIB_RSSeCall_t)(RSSLIB_RSSECmd_t *payload_addr);`

Table 8. RSSLIB_RSSeCmd_t description

Field name	Type	Comment
CmdID	uint32_t	Command ID
InputParamAddr	uint32_t	API input parameter address

3.3 RSSLIB & RSSe services

For each API, a command ID identifies the called RSSLIB or RSSe service. This command ID is the one mentioned as RSSLIB_CMD_ID or RSSe_KW_CMD_ID in the following figures.

Figure 3. Host basic RSSe service call

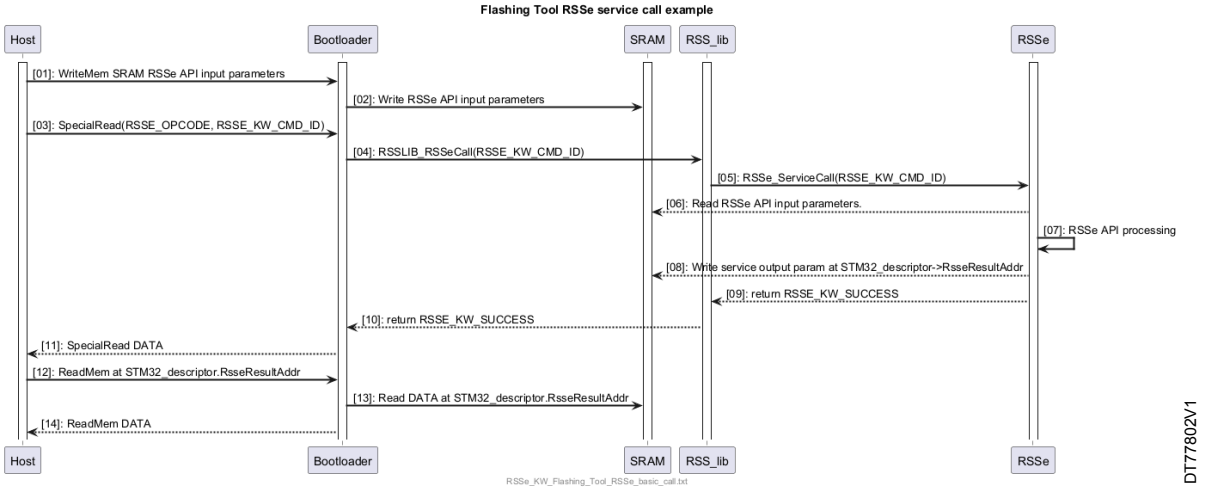
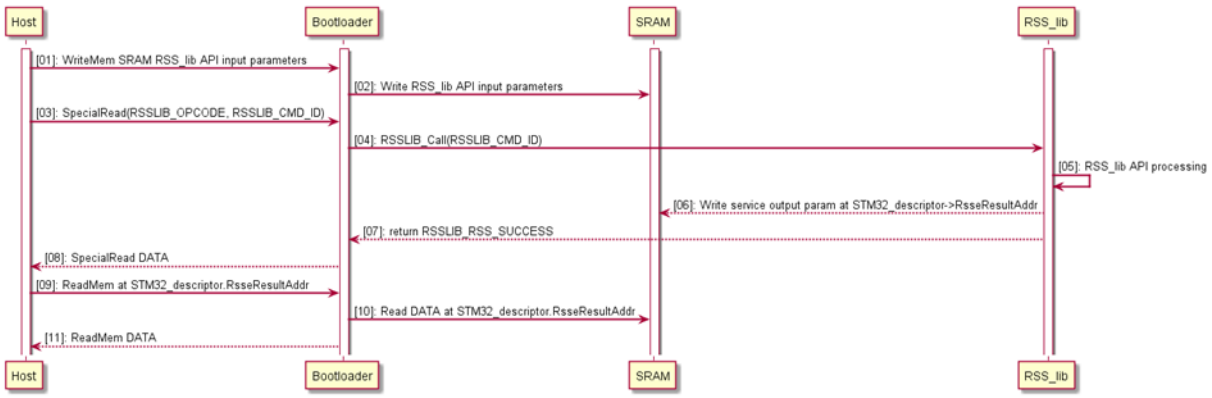


Figure 4. **Host basic RSS_lib service call**



3.3.1 RSSLIB_SetRSSCMD

Command ID: `RSSLIB_RSSCMD_CMD_ID` = 0x00

Service description: User calls RSSLIB_SetRSSCMD in order to request boot from root security service (RSS) on next boot. This request is canceled if bit BOOTLOCK within register FLASH_SECBOOTADD0R is already set.

Input parameter: uint32_t RSSCMD

Output parameter: None

Return value: `RSSLIB_RSS_SUCCESS`, `RSSLIB_RSS_ERROR`

Parameter description: see Table 9

Table 9. **uint32_t rsscmd values description**

Constant name	Value	Comment
RSSLIB_RSSCMD_RSSE_JTAG_BOOTLOADER	0xE00	Request RSS to install RSSe on next STM32 boot

3.3.2 RSSLIB_GetRssVersion

Command ID: `RSSLIB_GET_RSS_VERSION_ID` = 0x01

Service description: RSSLIB writes RSS version at address pointed by `STM32_descriptor→RsseResultAddr`

Input parameter: None

Output parameter: RSSLIB_RSSVersion_t description

Return value: `RSSLIB_RSS_SUCCESS`, `RSSLIB_RSS_ERROR`

Table 10. **RSSLIB_RSSVersion_t description**

Field name	Type	Comment
Patch	uint8_t	Bug fix
Minor	uint8_t	Backward compatible changes: public API deprecated improvements, for new API functions
Major	uint8_t	Backward incompatible changes: public API modification
Reserved	uint8_t	-

3.3.3 RSSe_KEYWRAP_GetVersion

Command ID: `RSSE_KEYWRAP_GET_VERSION_CMD_ID` = 0x02

Service description: RSSe writes RSSe KW version at address pointed by `STM32_descriptor->RsseResultAddr`.
Input parameter: None
Output parameter: RSSe_Version_t description
Return value: see RSSe_Version_t description.

RSSe_Version_t

Table 11. RSSe_Version_t description

Field name	Type	Comment
Patch	uint8_t	Bug fix
Minor	uint8_t	Backward compatible changes: public API deprecated improvements, for new API functions
Major	uint8_t	Backward incompatible changes: public API modification
Reserved	uint8_t	-

3.3.4 RSSe_KEYWRAP_GetStatus

Command ID: `RSSE_KEYWRAP_GET_STATUS_CMD_ID` = 0x01
Service description: RSSe writes KW status at `STM32_descriptor→RsseResultAddr`.
Input parameter: None
Output parameter: `uint32_t`, KW status is described in Table 12.
Return value: see RSSe KW status enumeration.

Table 12. RSSe KW status enumeration

Constant name	Constant value (C hexadecimal notation)
KW_STATUS_INITIALISED	0xEAEA0808U
KW_STATUS_NOT_INITIALISED	0xF5F50808U

3.3.5 RSSE_KEYWRAP_ECC_CHIP_PRIVATE_KEY_ID

Command ID: `RSSE_KEYWRAP_ECC_CHIP_PRIVATE_KEY_ID` = 0x06
Service description: This service returns the wrapped DUA private key container at `STM32_descriptor→RsseResultAddr`.
Input parameter: `InputKeyWrapEccChipPrivateKey_t`
Output parameter: `BinaryWrappedChipEccPrivateKey_t`
Return value: InputKeyWrapEccChipPrivateKey_t
The input parameters are detailed bellow in Table 13 to Table 19.

Table 13. InputKeyWrapEccChipPrivateKey_t

Field name	type	Comment
ECCChipPrivateKeyID	EccChipPrivateKeyID_t	ID of the chip private key
WrappingKeySel	WrappingKeySel_t	Selection of the key to wrap the chip private key
Reserved	uint8_t[1]	4 bytes alignment
Context	WrapUnwrapContext_t	Context to wrap chip private key
Reserved	uint32_t[10]	Reserved for future use

Field name	type	Comment
Crc	uint32_t	CRC on above fields

Table 14. EccChipPrivateKeyID_t

Constant name	Constant value (C hexadecimal notation)
KW_ECC_CPVK_DUA2_FU	0x00U
KW_ECC_CPVK_DUA2_LU	0x01U

Table 15. WrappingKeySel_t

Constant name	Constant value (C hexadecimal notation)
KW_WRAPPING_KEY_DHUK	0x00U
KW_WRAPPING_KEY_DHUK_XOR_BHK	0x01U

Table 16. WrapUnwrapContext_t

Field name	type	Comment
KeyUsage	KeyUsage_t	Indicate for what purpose the key is used
Reserved	uint8_t	Reserved for future use
SecAttr	SecAttr_t	Key used in secure or nonsecure domain
PrivAttr	PrivAttr_t	Key used in privileged or nonprivileged mode (only privileged mode available)
Reserved	uint8_t	Reserved for future use

Table 17. KeyUsage_t

Constant name	Constant value (C hexadecimal notation)
KW_KEY_ECC_USAGE_SIGN	0x00U
KW_KEY_ECC_USAGE_SCALAR_MUL	0x01U

Table 18. SecAttr_t

Constant name	Constant value (C hexadecimal notation)
KW_SEC_ATTR_SECURE	0x00U
KW_SEC_ATTR_NON_SECURE	0xFAU

Table 19. PrivAttr_t

Constant name	Constant value (C hexadecimal notation)
KW_PRIV_ATTR_PRIVILEGED	0x00U
KW_PRIV_ATTR_NON_PRIVILEGED	0xAFU (not supported)

The output container is composed of a header and a payload described in Table 20 to Table 24.

Table 20. **BinaryWrappedChipEccPrivateKey_t**

Field name	Type	Comment
Header	BinaryHeader_t	Header of the output container, containing elements to use the wrapped key.
Payload	EccWrappedChipPrivateKeyPayload_t	Payload of the output container, containing the wrapped key.

Table 21. **BinaryHeader_t**

Field name	Type	Comment
WrappedKeyID	WrappedKeyId_t	Identifier of the key.
PayloadSize	uint32_t	Payload size in bytes.
AdditionalData	AdditionalData _t	Data used to unwrap the wrapped key.
RSSeVersion	RSSe_version_t	RSSe KW version.
Reserved	uint32_t[7]	Reserved for future usage.
Crc	uint32_t	CRC on above fields and payload.

Table 22. **WrappedKeyId_t**

Field name	Type	Comment
ServiceId	uint8_t	Identifier of the key.
KeyUsageId	KeyUsage_t	Payload size in bytes.
KeySizeId	uint8_t	Key size identifier, 0x00 for ECC curve SECP256R1.
Reserved	uint8_t	4 bytes alignment.

Table 23. **AdditionalData _t**

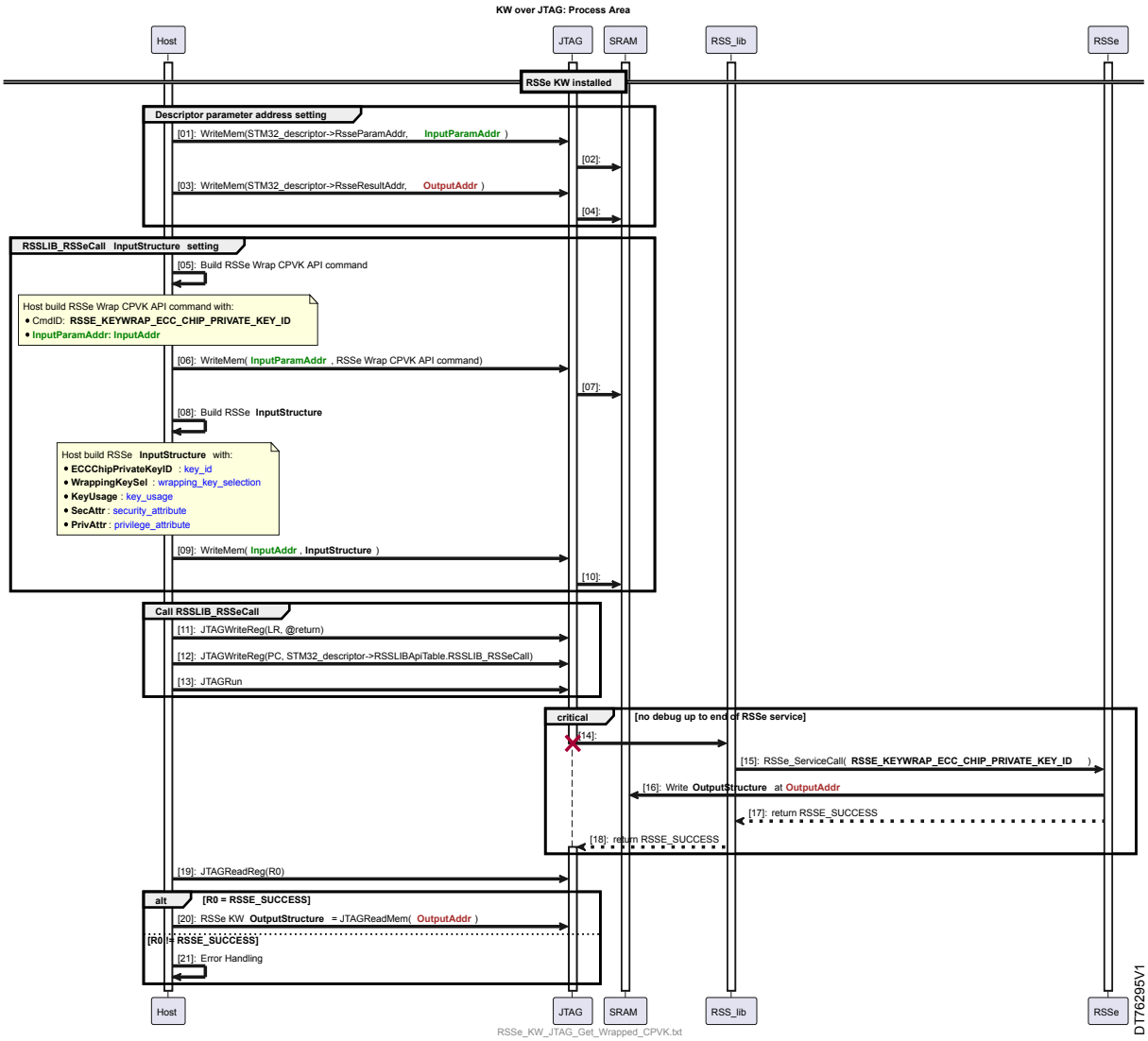
Field name	Type	Comment
IV	uint32_t[4]	Identifier of the key.
Tag	uint32_t[4]	Payload size in bytes.
AESMode	uint8_t	AES mode identifier, 0x02 for AES GCM
Reserved	uint8_t[3]	4 bytes alignment

Table 24. **EccWrappedChipPrivateKeyPayload_t**

Field name	Type	Comment
WrappedPrivateKey	uint8_t[32]	Wrapped Private Key

3.3.6 Dynamic description of key wrap command

Figure 5. **KW installation by JTAG: Get Wrapped CPVK**



3.4 Appendix A: kw.h file

```
/**
  ******************************************************************************
  * @file    kw.h
  * @author  MCD Application Team
  * @brief   key wrapping interface module.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */

#ifndef KW_H
#define KW_H
#include "rsslib.h"
```

```
/* Exported constants --------------------------------------------------------*/
/* KeyWrap error codes */
#define RSSE_KW_SUCCESS                      (RSSLIB_RSS_SUCCESS)
#define RSSE_KW_ERROR                        0xFFF4DA36U /* Generic value for error during key
 wrapping */
#define RSSE_KW_OB_INIT_ERROR                0xFFEA3D79U /* Wrong OB programmed for RSSe initi
alization */
#define RSSE_KW_SERVICE_INIT_ERROR           0xFFEAEE56U /* Error during service call */
#define RSSE_KW_CRC_ERROR                    0xF5F5E00EU /* CRC check error on input or output
 parameters */
#define RSSE_KW_NO_HW_CRYPTO                 0xF5F5A0AAU /* No HW crypto on this sample */
#define RSSE_KW_WRONG_INPUT_PARAM_CONTEXT    0xF5F50E0EU /* Context selected not supported */
#define RSSE_KW_WRONG_INPUT_PARAM_SELECTION 0xF5F5E0E0U /* key size ID or Curve ID not suppor
ted */
#define RSSE_KW_EXPORT_PUBLIC_KEY_ECC_ERROR 0xF5F58080U /* ECC export public key  error*/
#define RSSE_KW_WRONG_INPUT_PARAM_KEY_SEL    0xF5F58008U /* wrap AES key, wrapping key selecti
on not supported (only DHUK and DHUK XOR BHK supported for AES) */

/* Key sizes definition for output contener */
/* ECC SECP256R1 */
#define RSSE_ECC_256R1_CPVK_SIZEB 32U

/* Additionnal data */
#define RSSE_AES_GCM_IV_SIZEW 4U
#define RSSE_AES_GCM_TAG_SIZEW 4U
#define RSSE_AES_CBC_IV_SIZEW 4U

/* Exported types ------------------------------------------------------------*/
enum rsse_kw_service_id
{
  RSSE_KEYWRAP_SERVICE_ID_BASIC = (0U),
  RSSE_KEYWRAP_GET_STATUS_ID = (1U),
  RSSE_KEYWRAP_GET_VERSION_ID = (2U),
  RSSE_KEYWRAP_ECC_CHIP_PRIVATE_KEY_ID = (6U),
  RSSE_KW_MAX_ID
};

typedef enum
{
  KW_WRAPPING_KEY_DHUK = (0U),  /* Use DHUK to wrap */
  KW_WRAPPING_KEY_DHUK_XOR_BHK, /* Use DHUK ^ BHK to wrap */
} WrappingKeySel_t;

typedef enum
{
  AES_MODE_ECB = (0U), /* AES ECB Mode to wrap AES key, (no IV, no Tag) */
  AES_MODE_CBC,        /* AES CBC Mode to wrap AES key, (IV, no Tag) */
  AES_MODE_GCM         /* AES GCM mode to wrap asymmetric key, (IV, Tag) */
} AESMode_t;

typedef struct
{
  uint32_t IV[RSSE_AES_GCM_IV_SIZEW];   /* IV to unwrap the wrapped key */
  uint32_t Tag[RSSE_AES_GCM_TAG_SIZEW]; /* Tag used to verify the unwrapped key */
  AESMode_t AESMode;                    /* AES mode to wrap the key */
  uint8_t Reserved[3U];                 /* 4 bytes alignment */
} AdditionalData_t;

typedef enum
{
  KW_KEY_ECC_USAGE_SIGN = (0U), /* ECC Key used to compute signature */
  KW_KEY_ECC_USAGE_SCALAR_MUL,  /* ECC Key used to compute scalar multiplication */
} KeyUsage_t;

typedef enum
{
  KW_SEC_ATTR_SECURE = (0U),       /* Wrap key for usage in secure domain */
  KW_SEC_ATTR_NON_SECURE = (0xFAU) /* Wrap key for usage in non-secure domain */
```

```
} SecAttr_t;

typedef enum
{
  KW_PRIV_ATTR_PRIVILEGED = (0U),       /* Wrap key for usage in privileged domain */
  KW_PRIV_ATTR_NON_PRIVILEGED = (0xAFU) /* Wrap key for usage in non-privileged domain */
} PrivAttr_t;

typedef struct
{
  KeyUsage_t KeyUsage; /* Indicate for what purpose the key will be used */
  uint8_t    Hdpl;     /* HDPLevel using the key Parameter. Not used: 0xFE */
  SecAttr_t  SecAttr;  /* Key used in Secure or Non-Secure domain */
  PrivAttr_t PrivAttr; /* Key used in Privileged or Non-Privileged mode */
  uint8_t    CpuId;    /* On multi cores platform, indicates which CPU uses the key. Not used
: 0xFE */
} WrapUnwrapContext_t;

typedef enum
{
  KW_ECC_CPVK_DUA2_FU = (0U), /* Chip private key DUA2 FU */
  KW_ECC_CPVK_DUA2_LU         /* Chip private key DUA2 LU */
} EccChipPrivateKeyID_t;

typedef struct
{
  EccChipPrivateKeyID_t ECCChipPrivateKeyID; /* ID of the Chip Private key */
  WrappingKeySel_t      WrappingKeySel;      /* Selection of the key to wrap the Chip Private
 key */
  uint8_t               Reserved[1U];        /* 4 bytes alignment */
  WrapUnwrapContext_t   Context;             /* Context to wrap Chip Private key */
  uint32_t                 Reserved[10U];       /* Reserved for futur usage */
  uint32_t              Crc;                 /* CRC on above fields and key */
} InputKeyWrapEccChipPrivateKey_t;

/* Output contener structures */
typedef union
{
  EccChipPrivateKeyID_t EccCpvkSizeId;
} KeySizeId_t;

typedef struct
{
  uint8_t ServiceId;      /* Taken from rsse_kw_service_id values, only 0x06 available now */
  KeyUsage_t KeyUsageId;  /* Depending on the key type, indicates the usage of the key */
  KeySizeId_t KeySizeId;  /* From the union of the different key size identifiers */
  uint8_t Reserved;       /* 4 bytes alignment */
} WrappedKeyId_t;

typedef struct
{
  WrappedKeyId_t WrappedKeyID;      /* Identifier of the key */
  uint32_t PayloadSize;             /* Payload size in bytes */
  AdditionalData_t AdditionalData;  /* Data used to unwrap the wrapped key */
  RSSe_version_t RSSeVersion;       /* RSSe KW version */
  uint32_t Reserved[7U];            /* Reserved for future usage */
  uint32_t Crc;                     /* CRC on above fields and payload */
} BinaryHeader_t;

typedef struct
{
 uint8_t WrappedPrivateKey [RSSE_ECC_256R1_CPVK_SIZEB];
}EccWrappedChipPrivateKeyPayload_t;

typedef struct
{
  BinaryHeader_t Header;
  EccWrappedChipPrivateKeyPayload_t Payload;
} BinaryWrappedChipEccPrivateKey_t;
```

Revision history

Table 25. **Document revision history**

Date	Version	Changes
21-Mar-2025	1	Initial release.

Contents

List of tables

List of figures