TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC

About this document

Scope and purpose

The aim of this guide is to present the scope, the implementation, the algorithm and a demonstration of the **TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC** example code for the TLE989x Infineon Embedded Power ICs based on Arm® Cortex® M3. This example code can be found in the Keil µVision Pack Installer.

The full functionalities and characteristics of the embedded power devices are described in the datasheets and user's manual. Please refer to these documents for more detailed information. Furthermore, a low level (line-by-line) description of the code is not the aim of this document, although occasionally some codeblocks might be reported if necessary to the comprehension.

Note: The following information is given as a hint for the implementation of the system only and shall not be regarded as a description or warranty of a certain functionality, condition or quality of the referred devices or presented software example.

Intended audience

Design engineers, system engineers, embedded power designers

Table of contents

1 Introduction

Figure 1 shows the basic secure access mechanism implemented and the user api call sequence. The tool (user) must send random seed along with the secure access request service Id. The device will use the cryptolibrary to calculates its CMAC and sends positive response. In second step, tooll shall send the key request with CMAC value. The device will verify the recived data and provided either +ve or -ve response.

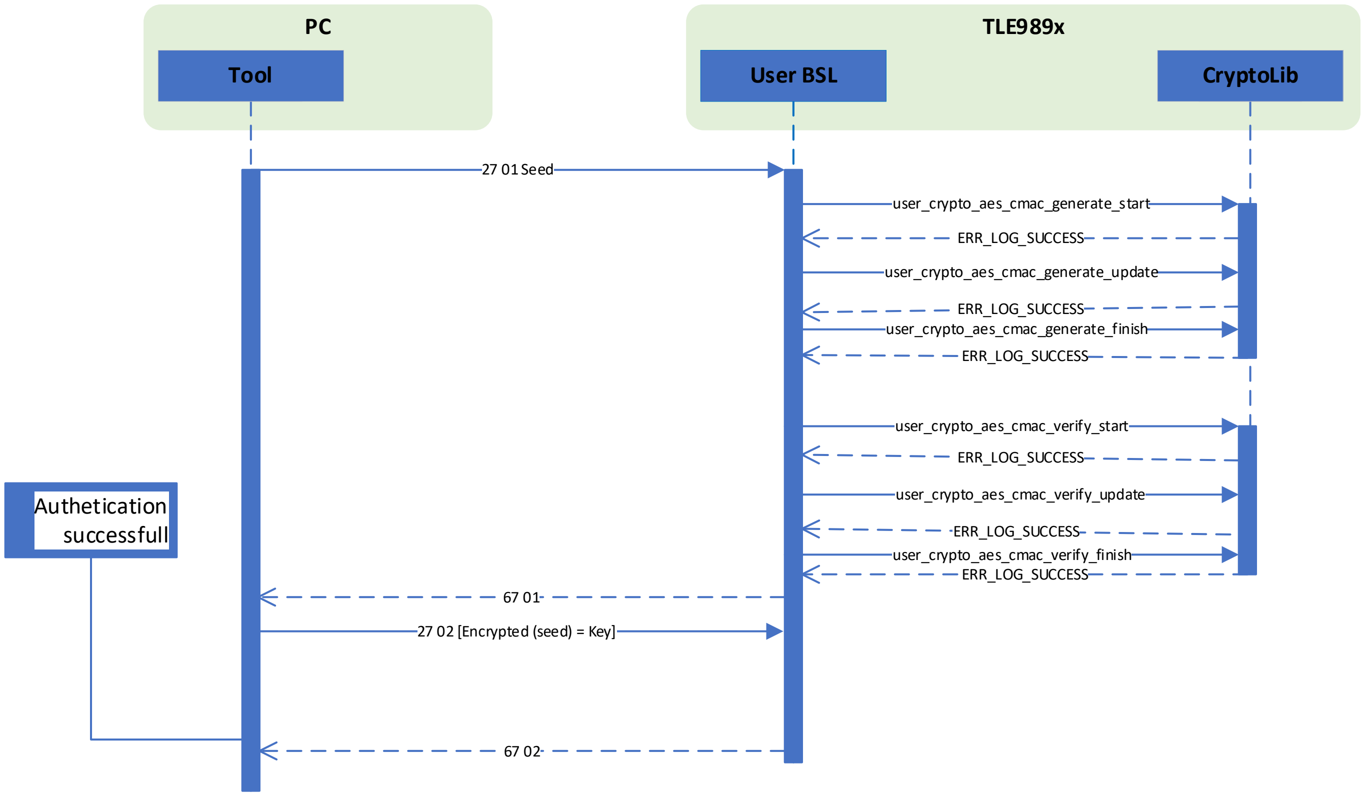
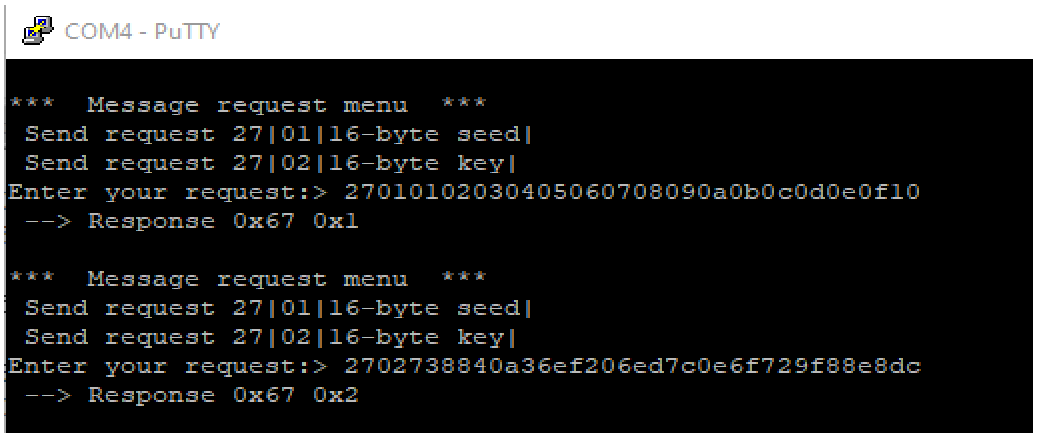

PC
Tool
TLE989x
User BSL
CryptoLib
27 01 Seed
user_crypto_aes_cmac_generate_start
ERR_LOG_SUCCESS
user_crypto_aes_cmac_generate_update
ERR_LOG_SUCCESS
user_crypto_aes_cmac_generate_finish
ERR_LOG_SUCCESS
user_crypto_aes_cmac_verify_start
ERR_LOG_SUCCESS
user_crypto_aes_cmac_verify_update
ERR_LOG_SUCCESS
user_crypto_aes_cmac_verify_finish
ERR_LOG_SUCCESS
Authetication successfull
67 01
27 02 [Encrypted (seed) = Key]
67 02

Figure 1 Secure access mechanism



```
***  Message request menu  ***
 Send request 27|01|16-byte seed|
 Send request 27|02|16-byte key|
Enter your request:> 2701010203040506070809a0b0c0d0e0f10
 --> Response 0x67 0x1

***  Message request menu  ***
 Send request 27|01|16-byte seed|
 Send request 27|02|16-byte key|
Enter your request:> 2702738840a36ef206ed7c0e6f729f88e8dc
 --> Response 0x67 0x2
```

Figure 2 uart output

Example:

Seed request: 2701010203040506070809a0b0c0d0e0f10

Key request: 2702738840a36ef206ed7c0e6f729f88e8dc

🟨 Service id 🟩 subfunction 🟦 16-byte hex encoded seed or encrypted key

Note:

- Encrypted key is CMAC of the seed (using the symmetric key stored in the device). The length of default symmetric key is 16 bytes.

  Default symmetric key: ffffffffffffffffffffffffffffffff

  The value of symmetric key (16 or 32 bytes) can be stored in NVM using the example SECURITY_WRITE_KEY_EXAMPLE_TLE989X.

- Complete access request string can be copy pasted to the Putty terminal and after pressing the “Enter” key sent command will be echoed to the terminal.

2 Hardware

This chapter shows how to run the TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC example with the TLE988x/TLE989x evaluation board. For this the project must be opened and compiled.

Figure 3 shows the TLE988x/TLE989x evaluation board. The application code must be loaded via a debugger (e.g. J-Link) to the board. The board must be powered with 12V (red and black connections).

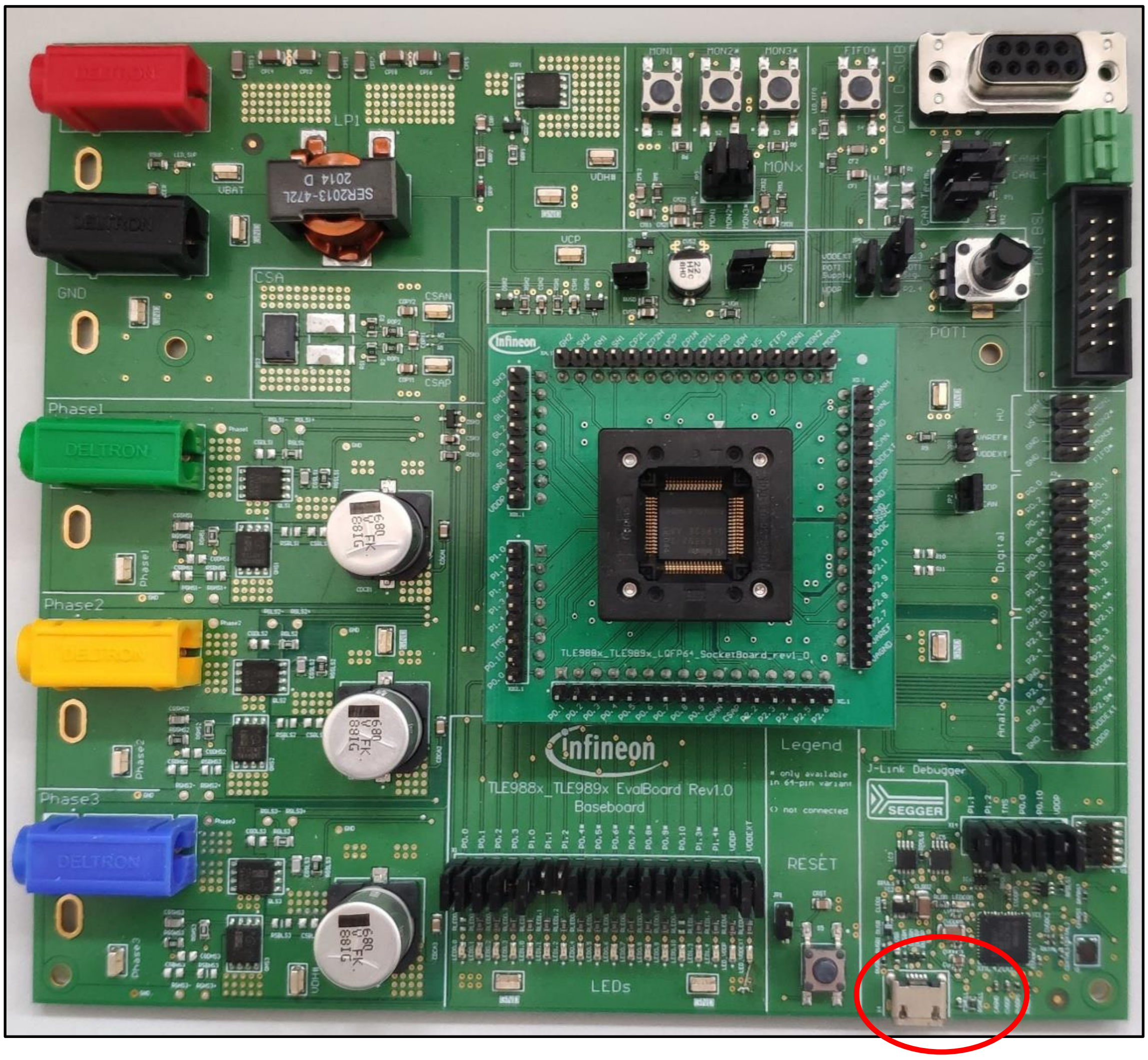


Figure 3 TLE988x/TLE989X evaluation board

Alternatively, a USB connection can be established to a local PC, which emulates a virtual COM port. The relevant COM device number can be identified via the Device Manager on Windows systems or the dmesg tool on Unix based operating systems.

In order to show the output on a command console, free tools like Putty or TeraTerm can be used. The UART1 in this example is configured with:

- a transmission baud rate of 115200,
- 8 data bits,
- 1 stop bit,
- no parity and no flow control.

3 Implementation

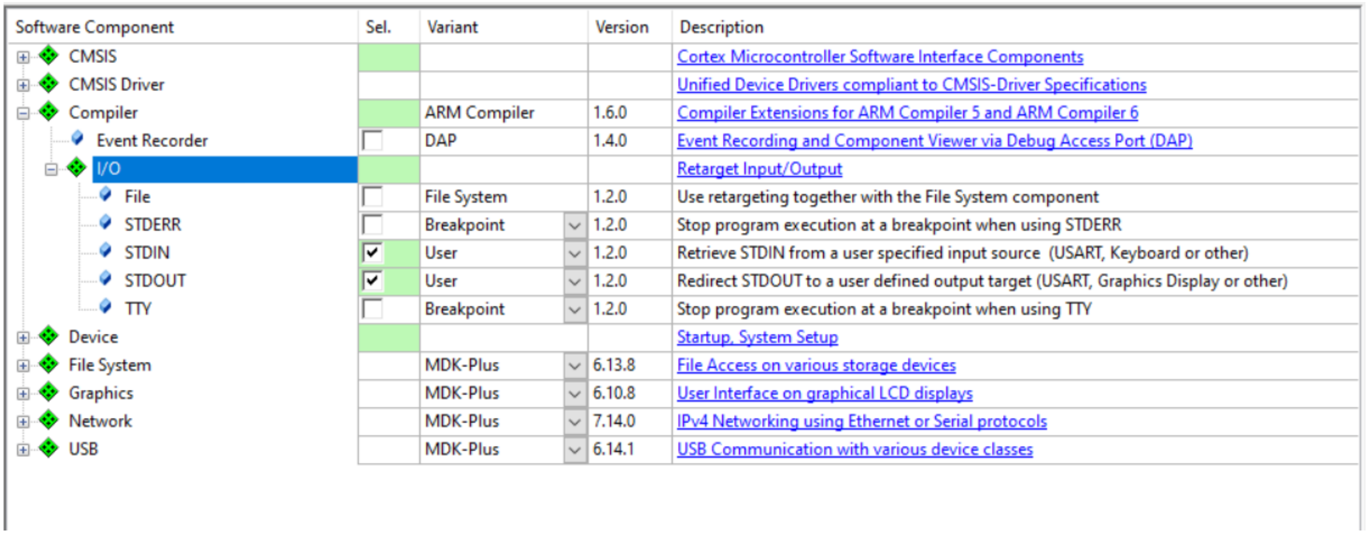
This chapter shows the process to follow to get a working secure access simple example.

3.1 Get the example via the Pack Installer for Keil

Open the Pack Installer within the Keil IDE.

Choose the appropriate device (here TLE9893_2QTW62S) on the left-hand side. On the right-hand side, select the tab Examples, where you can access the TLE98932QTW62S_SECURE_ACCESS_USING_CMAC example.

Clicking on “Copy” will copy the example on your computer and open it.



Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
Compiler		ARM Compiler	1.6.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Event Recorder	☐	DAP	1.4.0	Event Recording and Component Viewer via Debug Access Port (DAP)
I/O				Retarget Input/Output
File	☐	File System	1.2.0	Use retargeting together with the File System component
STDERR	☐	Breakpoint	1.2.0	Stop program execution at a breakpoint when using STDERR
STDIN	✓	User	1.2.0	Retrieve STDIN from a user specified input source (USART, Keyboard or other)
STDOUT	✓	User	1.2.0	Redirect STDOUT to a user defined output target (USART, Graphics Display or other)
TTY	☐	Breakpoint	1.2.0	Stop program execution at a breakpoint when using TTY
Device				Startup, System Setup
File System		MDK-Plus	6.13.8	File Access on various storage devices
Graphics		MDK-Plus	6.10.8	User Interface on graphical LCD displays
Network		MDK-Plus	7.14.0	IPv4 Networking using Ethernet or Serial protocols
USB		MDK-Plus	6.14.1	USB Communication with various device classes

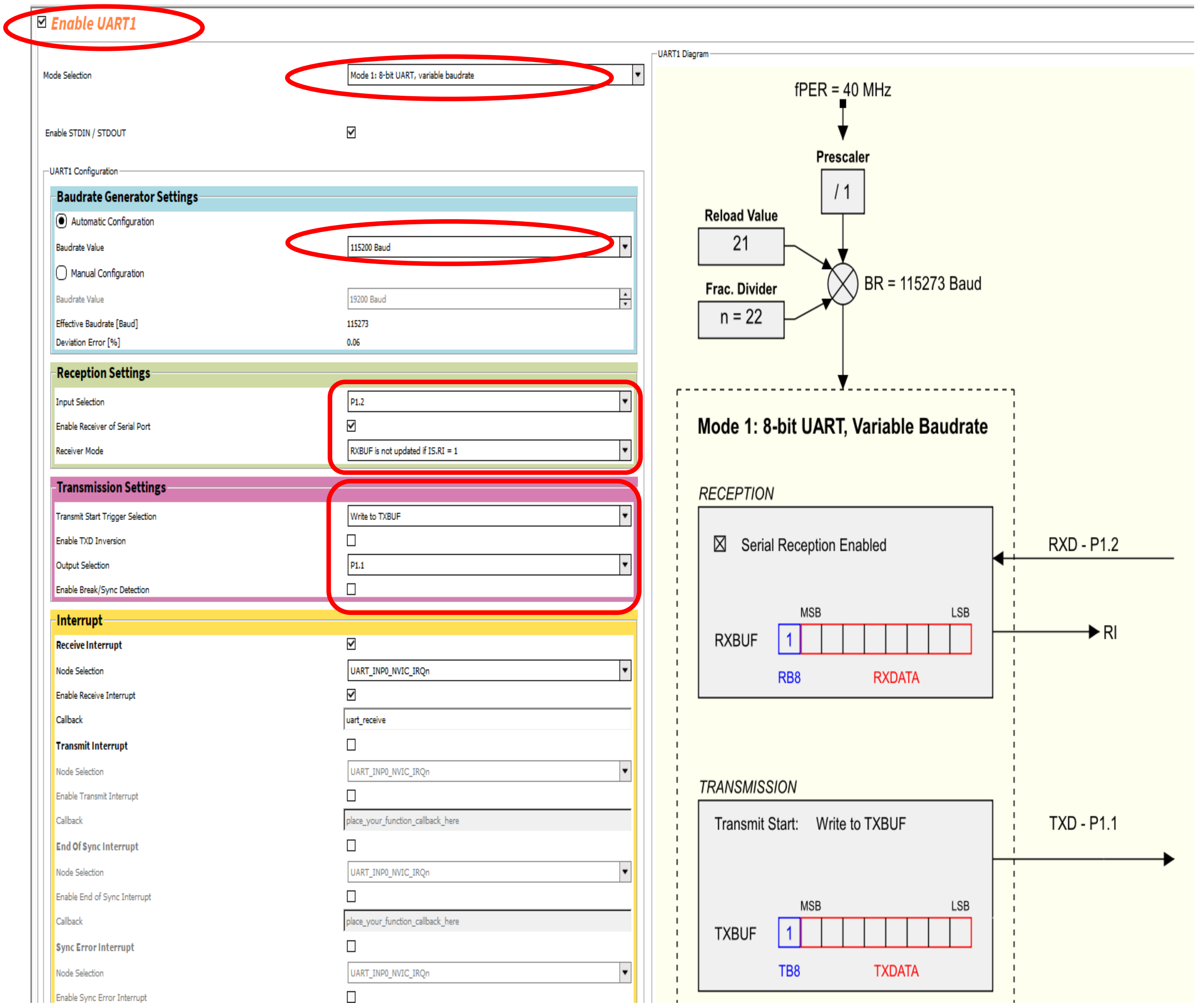
Figure 4 RTE settings for stdout and stdin

In order to redirect the stdout functions - the printf call in the example, adjust the runtime environment setting for the compiler within the Keil IDE. Select the option “User” under Compiler -> I/O -> STDOUT (see Figure 4).

In order to redirect the stdin functions - the stdin_getchar call in the example, adjust the runtime environment setting for the compiler within the Keil IDE. Select the option “User” under Compiler -> I/O -> STDIN (see Figure 4).

3.2 Configuration

In order to configure the UART module for the TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC example, select the UART tab. Enable the UART1 module. Next, select the 8-bit UART mode with variable baudrate. The baudrate is set to 115200 in the blue box Baudrate Generator Settings. This is one of the common speed settings for the UART. In the pink box Transmission Settings, select the pin P1.1. In the green box Reception Settings, select the pin P1.2 and set enable receiver of serial port radio box. In the yellow box Interrupt, enable both receive interrupt radio boxes and rewrite an intuitive name for the interrupt service routine, here uart_receive. See Figure 8 Config Wizard, module UART for more details



☑ Enable UART1

Mode Selection	Mode 1: 8-bit UART, variable baudrate
Enable STDIN / STDOUT	☑

UART1 Configuration

Baudrate Generator Settings

● Automatic Configuration	
Baudrate Value	115200 Baud
○ Manual Configuration	
Baudrate Value	19200 Baud
Effective Baudrate [Baud]	115273
Deviation Error [%]	0.06

Reception Settings

Input Selection	P1.2
Enable Receiver of Serial Port	☑
Receiver Mode	RXBUF is not updated if IS.RI = 1

Transmission Settings

Transmit Start Trigger Selection	Write to TXBUF
Enable TXD Inversion	☐
Output Selection	P1.1
Enable Break/Sync Detection	☐

Interrupt

Receive Interrupt	☑
Node Selection	UART_INP0_NVIC_IRQn
Enable Receive Interrupt	☑
Callback	uart_receive
Transmit Interrupt	☐
Node Selection	UART_INP0_NVIC_IRQn
Enable Transmit Interrupt	☐
Callback	place_your_function_callback_here
End Of Sync Interrupt	☐
Node Selection	UART_INP0_NVIC_IRQn
Enable End of Sync Interrupt	☐
Callback	place_your_function_callback_here
Sync Error Interrupt	☐
Node Selection	UART_INP0_NVIC_IRQn
Enable Sync Error Interrupt	☐

Figure 5 Config Wizard frequency configuration

Finally, save your configuration to take these changes into account (File -> Save).

3.3 Sample code flow

After setting the connection with PC using virtual COM via Putty for instance, upon reset device with send the uart message for sample commands format for secure access reuest. Then the sequence for secure access is as follows:

- First UART tool will send the secure access request id that is 0x27 0x01 (defined in UDS), along with 16 bytes of random seed data. The device validates the request ID and data length, ans passes this information to the cryptolibrary to calculate the CMAC. If every thing goes well it sends the positive response, else it will send the Negative Response Code (NRC).

- In second step, UART tool will send the key authentication request i.e (0x27 0x02), with CMAC value. Cryptolibrary verifies the key based on the received CMAC value, and provides the positive or negative response.

  All received NRCs (Negative Response Code) are based on the standard UDS protocol.

Figure 6 depicts the code flow implemented in the sample example. This is based on the UDS protocol.

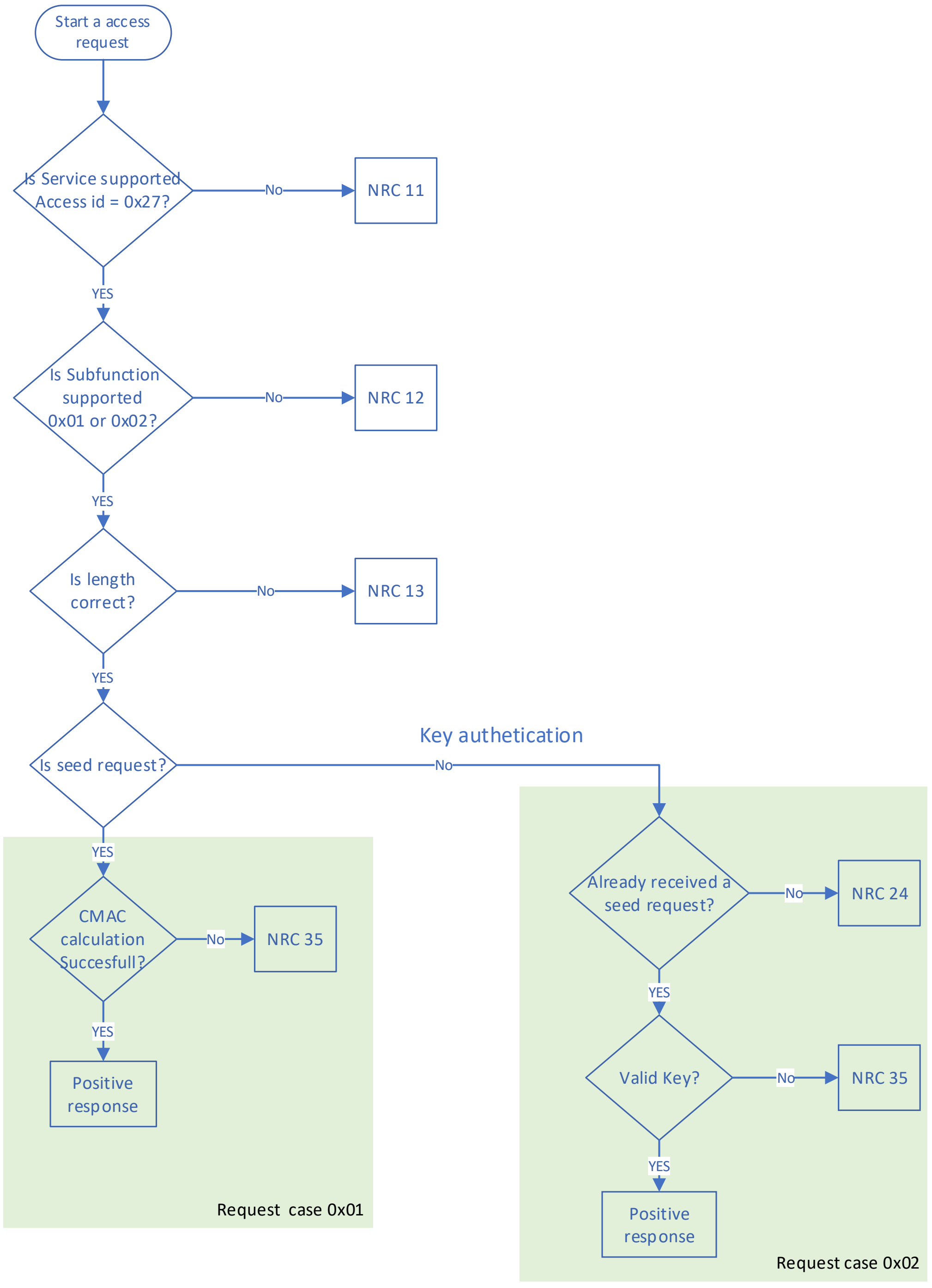


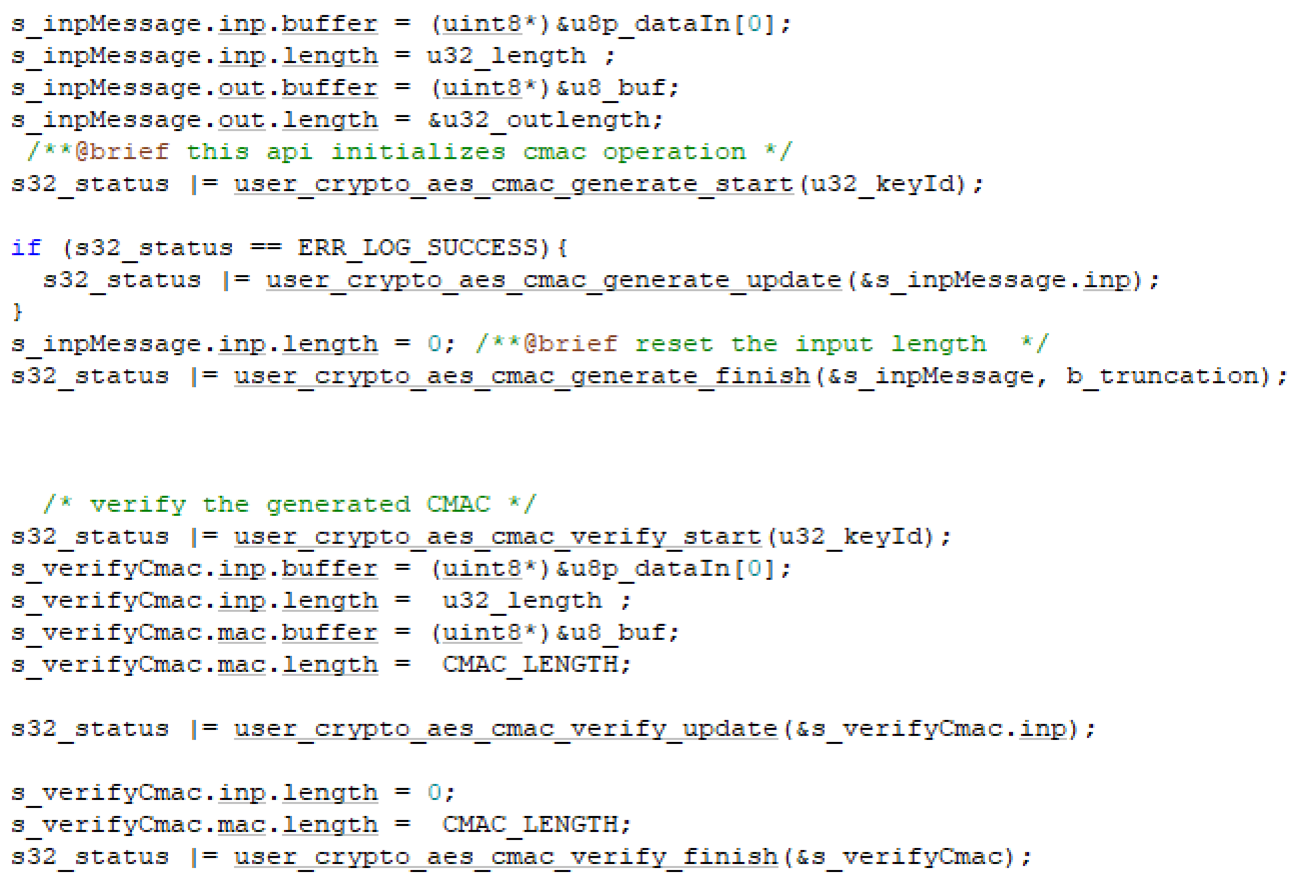
Figure 6 TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC application code

3.4 Sample code for AES CMAC user API

Figure 7 shows the application code of the user api usage.

`user_crypto_aes_cmac_generate_start` user API function initializes the CMAC generation with the specified AES key ID. `user_crypto_aes_cmac_generate_update` user API function updates an ongoing CMAC generation with the specified buffer. `user_crypto_aes_cmac_generate_finish` user API function finalizes an ongoing CMAC verification with the specified MAC.

`user_crypto_aes_cmac_verify_start` API function initializes the CMAC verification with the specified AES key ID. `user_crypto_aes_cmac_verify_update` API function updates an ongoing CMAC verification with the specified buffer. `user_crypto_aes_cmac_verify_finish` API function finalizes an ongoing CMAC verification with the specified MAC.



```c
s_inpMessage.inp.buffer = (uint8*)&u8p_dataIn[0];
s_inpMessage.inp.length = u32_length ;
s_inpMessage.out.buffer = (uint8*)&u8_buf;
s_inpMessage.out.length = &u32_outlength;
 /**@brief this api initializes cmac operation */
s32_status |= user_crypto_aes_cmac_generate_start(u32_keyId);

if (s32_status == ERR_LOG_SUCCESS){
  s32_status |= user_crypto_aes_cmac_generate_update(&s_inpMessage.inp);
}
s_inpMessage.inp.length = 0; /**@brief reset the input length  */
s32_status |= user_crypto_aes_cmac_generate_finish(&s_inpMessage, b_truncation);



  /* verify the generated CMAC */
s32_status |= user_crypto_aes_cmac_verify_start(u32_keyId);
s_verifyCmac.inp.buffer = (uint8*)&u8p_dataIn[0];
s_verifyCmac.inp.length =  u32_length ;
s_verifyCmac.mac.buffer = (uint8*)&u8_buf;
s_verifyCmac.mac.length =  CMAC_LENGTH;

s32_status |= user_crypto_aes_cmac_verify_update(&s_verifyCmac.inp);

s_verifyCmac.inp.length = 0;
s_verifyCmac.mac.length =  CMAC_LENGTH;
s32_status |= user_crypto_aes_cmac_verify_finish(&s_verifyCmac);
```

Figure 7 TLE9893_2QTW62S_SECURE_ACCESS_USING_CMAC application code

References

See the code examples at **www.infineon.com**

Revision history

Document version	Date of release	Description of changes
1.0	2021-11-04	Initial version
1.1	2022-10-13	Editorial changes

Edition 2022-10-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference