AN12324

LPC55Sxx Usage of the Physically Unclonable Function and Hash Crypt to AES Coding

Rev. 2 — 22 September 2023 Application note

Document Information

Information	Content
Keywords	AN12324, physically unclonable function, key management, security feature, LPC55Sxx, LPC55
Abstract	This application note describes how to securely generate, store, and retrieve user keys using the root key.

1 Introduction

This application note describes how to generate, store, and retrieve user keys securely using the root key. The root key is used as a key encryption key (KEK) to protect other user keys. The physically unclonable function (PUF) peripheral generates the root key. The PUF technology generates a device-unique 256-bit KEK, which is a digital fingerprint of a device. The encrypted keys (key codes) can then be stored in the MCU.

This document discusses the key code stored in the protected flash region (PFR) flash memory and key provisioning to the system via the in-system programming (ISP) and in-application programming (IAP) commands.

This application note also describes the following communication tools:

- blhost
- elftosb

Note: *This application note was written using legacy tools and flow. The MCUXpresso Secure Provisioning (SEC) Tool and SPSDK are the latest tools. The information in this application note describing the secure flow within the chip still applies, but we recommend using the latest tools (MCUXpresso SEC or SPSDK) instead of following the steps shown here. For any questions, please contact your local support.*

2 Acronyms

Table 1 lists the acronyms used in this document.

Table 1. Acronyms

Acronym	Meaning
CMPA	Customer manufacturing programmable area
UDS	Unique device secret
KC	Key code
API	Application programming interface
UART	Universal asynchronous receiver-transmitter
I2C	Inter-Integrated Circuit
SPI	Serial peripheral interface
USB	Universal serial bus
HID	Human interface device
SB	Secure bootloader
RAM	Random access memory
SDK	Software development kit
COM	Communication port
ECB	Electronic code book

3 Physically unclonable function

This section explains in detail about PUF.

3.1 PUF features

The PUF has the following features:

- The PUF peripheral generates a 256-bit key. The key is a device-unique, unclonable, and used as the KEK, which is a digital fingerprint.
- The PUF can generate 64-bit to 4096-bit keys.
- The PUF encrypts a 64-bit to 4096-bit key to a key code.
- The PUF decrypts a key code back to a key.
- The key can be sent to the hash crypt engine (or PRINCE) through the internal hardware bus.
- The CPU reads the key through the register and the AHB bus.
- The PUF offers other features to prevent attacks. For example, by blocking functionalities (enroll, code output, keylock) or by offering enhanced side-channel protection (by using a keymask).

3.2 PUF unique key principle

The uniqueness of the KEK generated by the PUF is based on variations in the attributes of transistors when chips are fabricated (length, width, thickness). Each time the SRAM block powers on, the cells are either 1 or 0. The startup values create a random and repeatable pattern, which is unique to each chip and is called SRAM startup data (SD).

The SRAM SD, along with the activation code (AC), are turned into a digital fingerprint. This digital fingerprint is used as a secret key that builds the foundation of a security subsystem. This digital fingerprint is also used as a root key or KEK.

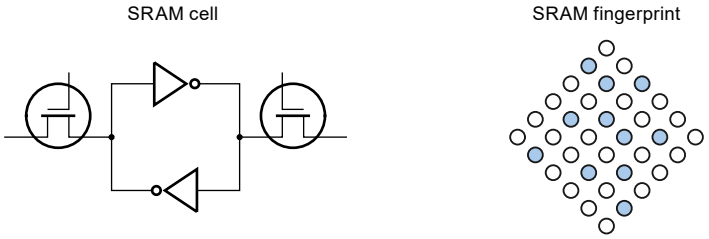

Figure 1. PUF principle based on an array of SRAM cells

The PUF module features embedded error correction, therefore in the worst-case scenario, the probability of a key reconstruction failure is < 10^{-9}.

3.3 PUF usage

The security of any system depends on how securely the keys are stored. If the key is placed in the inaccessible flash memory, an adversary can uncap the chip and read the memory. The same applies for the fuses, which can be read after uncapping.

When you encrypt a key, you must store it somewhere. If adversaries read out the whole flash, they can clone your device because the stored key is the same for the whole production. If adversaries successfully crack your key, the key is valid for all your devices. All these problems are solved using PUF.

Each MCU has its own unique digital fingerprint, which is not stored on the chip and cannot be read when the device is not powered. The following sections show how to use the PUF to generate, store, and retrieve keys.

The PUF controller functionality includes the following commands:

- `Enroll`
- `Start`
- `SetKey`
- `GenerateKey`
- `GetKey`
- `Zeroize`

- DisableEnroll
- DisableSetKey

These commands are used to control the PUF key management.

3.3.1 PUF enroll command

To start working with PUF, you must enroll it. During enrolling, the SRAM SD is derived to a digital fingerprint and a corresponding AC is generated. The AC is read out through the CODEOUTPUT register.

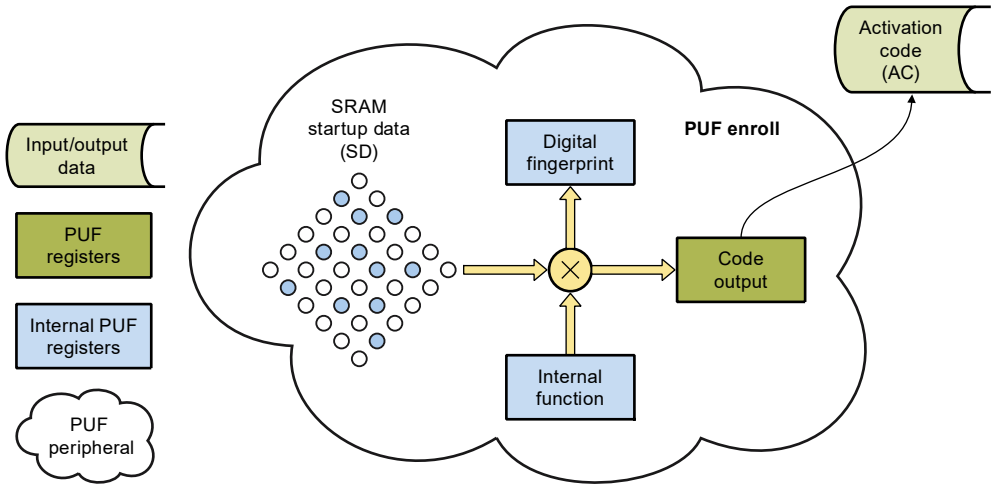

Figure 2. PUF enroll command

Each time the PUF Enroll is performed, a new and different AC and digital fingerprint is generated. The key code can be decrypted to a key with a corresponding digital fingerprint used for the key encryption to the key code. Therefore, to decrypt the key code later, the AC must be stored. The AC is a 1192 byte data chunk, which must be stored into the non-volatile memory.

After a successful PUF Enroll command, the SetKey command becomes available. The PUF must be power-cycled and started to enable the GetKey command. The keys and the AC can also be generated using the ISP commands and stored into the flash. The ISP commands use the ROM code to call the PUF functions internally.

3.3.2 PUF start command

After the PUF Enroll command is performed and the AC is stored, the PUF can be used. The PUF Start command must be used to power cycle and start the PUF. The AC is loaded into the PUF through the CODEINPUT register. The PUF engine combines it together with the SRAM SD into a digital fingerprint.

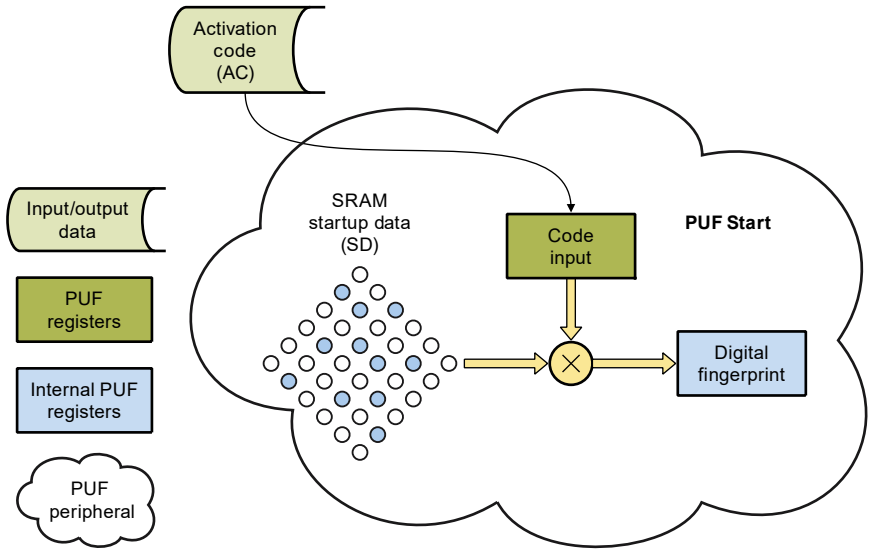

Figure 3. PUF start command

The digital fingerprint is a 256-bit key used as a root key or a KEK for encryption/decryption of user keys. After a successful `Start` command, the `SetKey` and `GetKey` commands become available. The digital fingerprint is available in the PUF until the next power-cycle.

3.3.3 PUF SetKey command

The `SetKey` command is used to create a user key and a corresponding key code. The user key and the digital fingerprint are combined to generate a key code.

The input parameters of the `SetKey` command are as follows:

- *User key*: The user key is sent to the PUF through the `KEYIN` register and must have a correct length, depending on the key size.
- *Key size*: The key size is sent to the PUF through the `KEYSIZE` register. It must have a number between 0 and 63. It instructs the PUF peripheral to generate a corresponding key size between 64 bits and 4096 bits. 0 generates a 4096-bit key. For more information, see the product-specific reference documentation.
- *Key index*: The key index refers to the index of where to set the key in the key array. The key index is a number from 0 to 15, which is added to the header of a generated key code. It is sent to the PUF through the `KEYIDX` register. The key index is retrieved back during the `GetKey` command and accessible in the `KEYOUTIDX` register.

As an exception, the `GetKey` command can be instructed to send the decrypted key to the AHB or one of the peripherals (AES or PRINCEx) through the internal secure bus. The `GetKey` command can be instructed using a key index of 0. The key code is a secure representation of the encrypted key. This key code must be read out through the `CODEOUTPUT` register and safely stored into the flash memory. Another technique blocks a key with a given index from appearing on the APB register interface and reading by the CPU. This setting can be configured using one or more `IDXBLK` registers and their duplicates (`IDXBLK_DP`). For more details, refer to the product-specific reference documentation.

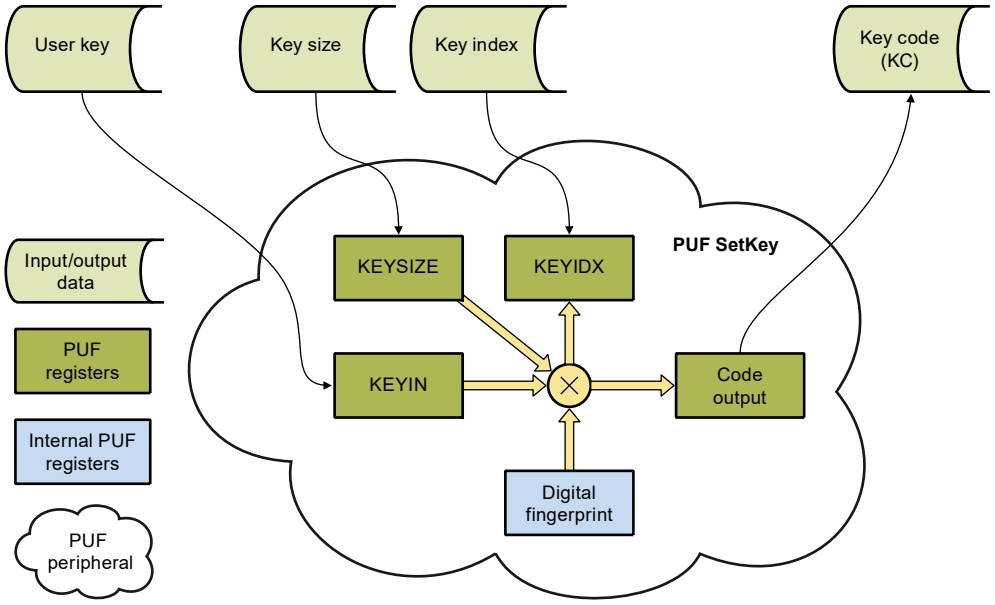

Figure 4. PUF SetKey command generating the user key code

The generated key code is a block of data with a 32-bit key header in the beginning.

The 32-bit key header has the following features:

- *Type*:
  - 0: Generated key
  - 1: User key
- *Index*: From 0 to 15. "0" is a special case to send the key internally to the AES or PRINCE engines. Otherwise, it can be used as a tag of the user.
- *Size*: From 0 to 63 for keys ranging from 64 bits to 4096 bits (0 means 4096 bits).

3.3.4 PUF GenerateKey command

The `GenerateKey` command produces a random number using the PUF peripheral, while the `SetKey` command allows the user to define and provide the key. The only required parameter (key size) must be entered into the `KEYSIZE` register and the key index must be entered into the `KEYIDX` register. The output is read from the `CODEOUTPUT` register and must be stored for the key reconstruction later on.

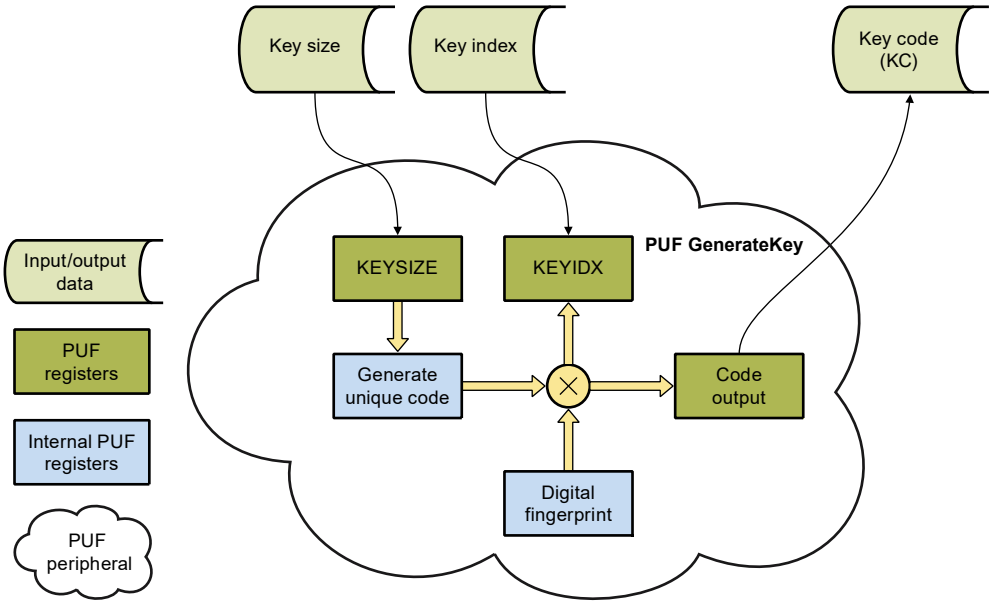

Figure 5. PUF GenerateKey command generating the key code

3.3.5 PUF GetKey command

The `GetKey` command is used to get a key out of a key code. The key code is input into the `CODEINPUT` register after the PUF is started with a corresponding AC.

Note: *The `GetKey` command is not available after the `Enroll` command.*

After combining the key code with the digital fingerprint, the key is retrieved. The key is sent to the internal secret hardware bus (key index = 0) or the `CODEOUTPUT` register (key index > 0) depending on the value of the key index in the key code header. The user code must read the key from this register and the key index from the `KEYIDX` register.

The internal hardware bus can feed the key to hardware encryption units AES, PRINCE1, PRINCE2, or PRINCE3, depending on the setting of the `KEYENABLE` register.

The `KEYMASK [0..3]` register must be loaded with random values. It is used as a mask for the keys stored in the key hold registers. Masking these registers obscure the keys, which are a countermeasure against side-channel attacks.

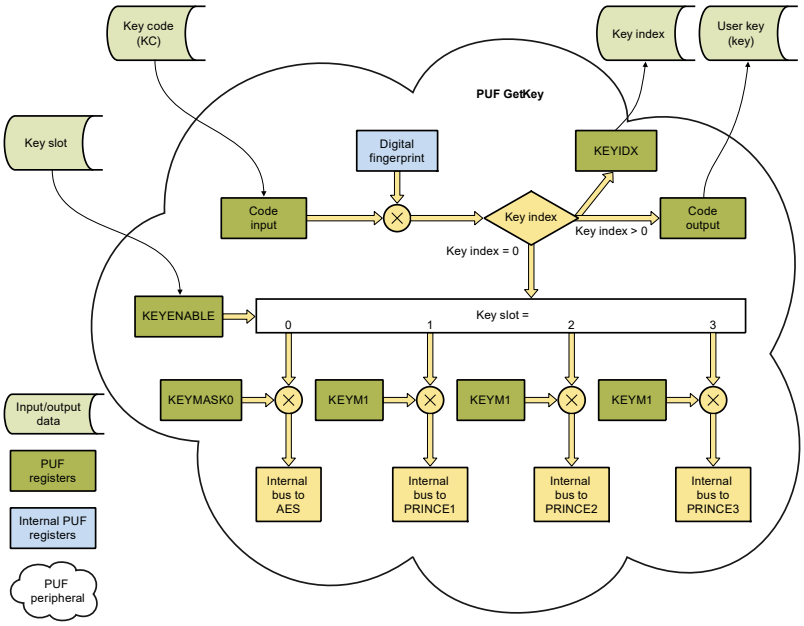

Figure 6. PUF GetKey command to get a key out of a key code

3.3.6 PUF zeroize command

When the `Zeroize` command is called, all the internal, critical, and security parameters are erased and the PUF controller goes to the error state. The device must be switched back on before performing any further operations.

3.3.7 PUF DisableEnroll command

Calling the `DisableEnroll` command causes the PUF peripheral to block the enroll operation.

3.3.8 PUF DisableSetKey command

The `DisableSetKey` command blocks the key output data as follows:

- If the bit is set to 1, the PUF peripheral blocks the key output (keyout data = 0) when the key code index = 15.
- If the bit is set to 0, the key output is not blocked even when the key output index = 15.

4 Key management

This section explains how to do key management by using the blhost and elftosb-gui tools.

4.1 Key storage in the PFR

Section 3 describes that a key code can be stored securely into the MCU flash memory or any other external memory, because the master key is not physically accessible on the MCU. The place to store the key codes is in the PFR on the MCU.

The PFR is protected from being accessed from the bus and has two areas as follows:

- *Customer Field Programmable Area (CFPA)*: This area holds the monotonic counters, key revocation, and PRINCE IV codes.

- *Customer Manufacturing Programmable Area (CMPA)*: This area holds the boot configurations, security policies, user-defined data, and PUF key storage.

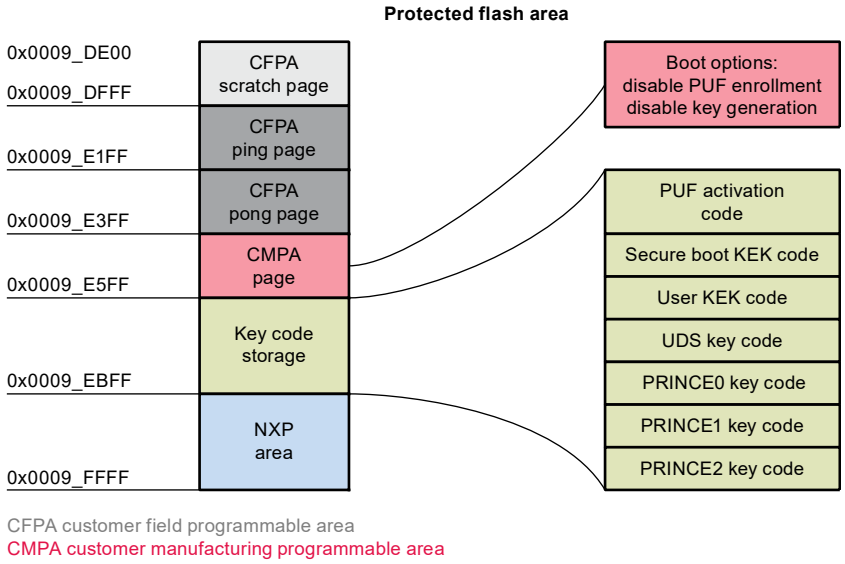

Figure 7. Keys and PUF boot option in the PFR

Note: *The boot code uses and manages the key storage.*

The secure boot uses the secure boot KEK code. The flash encryption by PRINCE uses the PRINCE1 to PRINCE3 key codes for its protected regions. When the correct key codes are found, the boot process applies them to PRINCE. The boot process checks the key code storage for existing code validity. If the AC or KC is corrupt, the booting stops and blocks the MCU forever.

This region is also protected against data corruption or modification. When locked by the ROM, all regions are protected from the erase/write commands. Each region has an SHA256 hash digest field used in the deployment Life-Cycle state to cross-check the integrity of the region.

Figure 7 shows the positions for key codes in the PFR memory. There is enough place for the AC and five key codes. These five key codes include secure boot key code, user key code, which can be used for AES, and three PRINCE key codes. One position is secured for the unique device secret (UDS). Each key code position has the maximum size limited to 52 bytes, which gives the maximum key size of 256 bits. This size can seem low, but the internal encrypting engines accept a maximum key size of 256 bits. The AES maximum allowed key size is 128/196/256 bits. The maximum PRINCE allowed key size is 128 bits.

From the PUF peripheral use point of view, the important part is also placed in the CMPA page. There are the "PUF Enrollment Disable" and "PUF Code Generation Disable" options. These options are copied to the PUF peripheral during the boot process. Remember that these settings can be changed only after a power cycle. If the options are loaded during boot, the PUF can never be enrolled or another key code can never be generated in the user code. However, you can read the existing AC and key codes from the PFR and use them.

The CFPA region can be updated through the ROM API, both ISP and IAP. The ISP function supports the key-provisioning commands to generate the key codes and store them into the PFR. Another option is to use the flash IAP commands to manage the key storage area in the PFR from the user code.

The following sections describe the IAP and the ISP commands dedicated to work with the key storage area.

4.2 ISP key provision commands

The ISP is a bunch of functions, which support image programming through serial interfaces such as, UART, I^2C, SPI, USB HID and so on. The `KeyProvision` is a security-related command to install the pre-shared keys, generate random keys, and save them into the PFR key store.

The three possible parameters for the `KeyProvision` command are as follows:

- *Key operation*: Required to specify the `KeyProvision` command behavior and can have the following values:
  - `Enroll`
  - `SetUserKey`
  - `SetIntrinsicKey`
  - `WriteNonVolatile`
  - `ReadNonVolatile`
  - `WriteKeyStore`
  - `ReadKeyStore`
- *Key type*: This parameter defines what the generated key code is used for.

Key type	Value
User KEK	11
SB KEK	3
PRINCE0	7
PRINCE1	8
PRINCE2	9
UDS	12

- *Key size*: The key size parameter is defined in bytes.

4.3 Key store area configuration through the blhost PC application

The blhost[1] application is used on a host computer to issue commands to an MCU running an implementation of the MCU bootloader. The MCU bootloader and the blhost application work together to enable programming of firmware applications into the MCU device without the use of a programming tool.

The blhost application also supports several key-provisioning-related commands described in the following examples:

- `Enroll PUF`: This command calls a ROM code that configures, starts, and enrolls the PUF peripheral:

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning enroll
```

- `Generate UDS`: This command calls the ROM code that uses the PUF to generate an intrinsic 32-byte key and its key code. The command stores them to the CMPA key code storage `[UDS key code]` position (RAM version):

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_key 12 32
```

- `Generate user-secure bootloader code SB KEK`: This command calls the ROM code that generates a key code based on the key stored in the `tempSbkek.bin` file. This command stores it to the CMPA key

1 Both the blhost and elftosb-gui can be found on the MCUBOOT webpage.

code storage `[secure boot KEK]` position (RAM version), where `tempSbkek.bin` is a plain text file with the SB KEK:

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_user_key 3 ".\temp
\tempSbkek.bin"
```

- `Generate user key code SB KEK`: This command sets the user key code and stores it to the CMPA key code storage `[user key code SB KEK]` position (RAM version), where `tempSbkek.bin` is a plain text file with the SB KEK:

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_user_key 11 ".\temp\UserKek.bin"
```

- `Generate intrinsic key code for PRINCE0`: This command generates a 128-bit intrinsic key code for PRINCE0 and stores it to the CMPA key code storage `[PRINCE0 key code]` (RAM version):

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_key 7 16
```

- `Generate intrinsic key code for PRINCE1`: This command generates a 128-bit intrinsic key code for PRINCE1 and stores it to the CMPA key code storage `[PRINCE1 key code]` (RAM version):

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_key 8 16
```

- `Generate 16 bytes intrinsic key code for PRINCE2`: This command generates a 128-bit intrinsic key code for PRINCE2 and stores it to the CMPA key code storage `[PRINCE2 key code]` (RAM version):

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning set_key 9 16
```

- The following command is used to update the key storage in the PFR with the RAM version of key codes created in the previous operation. These codes include the AC created during the enroll operation:

```
.\blhost\win\blhost.exe -V -p COM32,57600 -- key-provisioning write_key_nonvolatile 0
```

4.4 Key provisioning through the elftosb-gui tool

A window-based PC tool called elftosb-gui[2] is built on the command-line blhost application that aids in creating an image and adds all necessary settings and encryption. The tool can be used to tell the PUF to enroll and store an AC, generate a key/key code, and store it to the PFR.

Note: *Currently, the tool does not support the generation of a user KEK code.*

2 Both the blhost and elftosb-gui can be found on the MCUBOOT webpage.

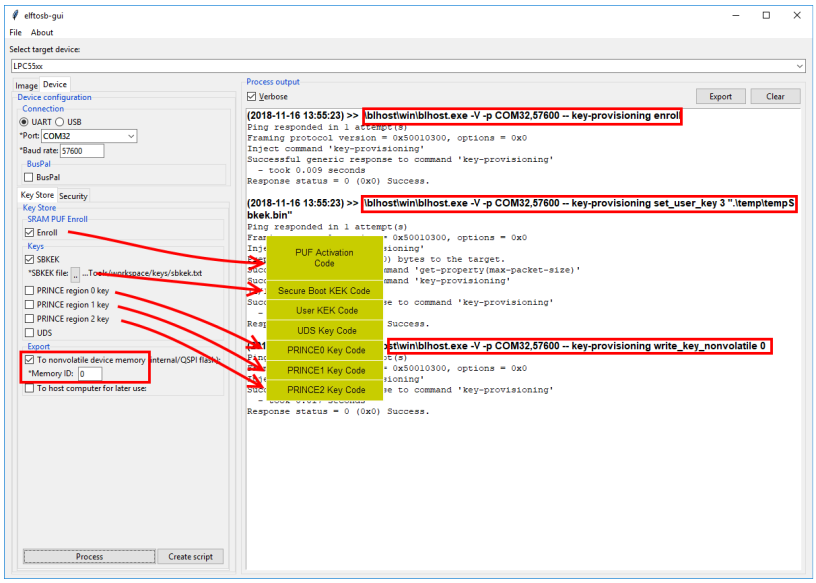

Figure 8. Key provisioning through the elftosb-gui tool

The elftosb tool can also set the security options of the PUF, disable the `Enroll` command, and disable the `SetKey` command.

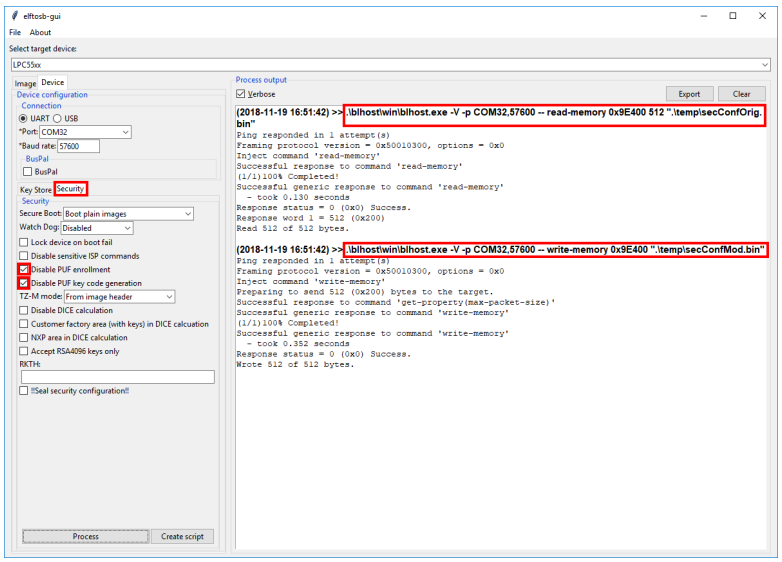

Figure 9. Various security settings supported by the elftosb tool

4.5 Key management IAP commands

The key codes can be programmed into the key storage area of the PFR in the MCU by the user application. Several FFR IAP ROM commands are dedicated to read the key codes, AC, and to store a block of data into the key storage area.

Start the PUF with the AC that has been used to create the key codes. These key codes are stored in the PFR key storage area.

The key management IAP commands are as follows:

- The `ffr_keystore_get_ac` returns the AC from the key storage:

```
status_t ffr_keystore_get_ac (flash_config_t *config, uint8_t* pActivationCode)
```

  Where:
  – `pActivationCode` is a pointer to a buffer long enough to hold a 1192 byte AC.

The AC must be passed to the PUF using the PUF `Start` command.

- The `ffr_keystore_get_kc` is used to load the key code from the PFR:

```
status_t ffr_keystore_get_kc (flash_config_t *config, uint8_t* pKeyCode, ffr_key_type_t keyIndex);
```

  Where:
  – `pKeyCode` is the buffer into which the key code is copied.

The key types are specified in Table 2.

Table 2. Key types

Key type	Value
SB KEK	0
User KEK	1
UDS	2
PRINCE0	3
PRINCE1	4
PRINCE2	5

The PFR non-volatile memory can be read using commands such as `ffr_keystore_write`, `ffr_infield_page_write`, and `ffr_get_customer_infield_data`.

For more details, see the product-specific reference documentation.

5 PUF testing software usage

The software package `AN12324SW.zip` accompanies this document, which can be downloaded from the NXP website.

Note: *The following chapter/code example is for the LPC55S69 variant.*

This software enables you to perform the following commands:

- Basic PUF commands: `Enroll`, `Init`, `Start`, `Stop`, `SetKey`, and `GetKey`
- Security-enhancing commands: `DisableEnroll` and `DisableSetKey`

The software also enables you to get the key code and then provides this key code to the AES module. The key code is provided through a standard memory bus or a secret bus between the PUF and the hash crypt engine. You can also encrypt/decrypt a 16-byte data block with the provided key. The example code is based on the SDK and uses the SDK drivers.

The software is controlled via a serial line terminal and uses the onboard built-in virtual serial COM port (LPC-LinkII UCom port). Use any ASCII-based serial COM port terminal application, such as Tera Term or Putty. The terminal must have local echo enabled.

5.1 PUF example code

To set up the PUF example code, perform the following steps:

1. Import the software package into **MCUXpresso** and flash it to the board. For more information about the importing and flashing project in MCUXpresso, see the MCUXpresso-IDE.
2. Connect the LPCXpresso5500 board to the PC via the onboard USB header P6. The **LPCLinkII CMSIS DAP** and the virtual serial **COM port LPC-Link UCOM** appear in the Windows OS device manager.
3. Use the terminal application of choice to open a correct COM port and set the parameters to "115200 bit/s, bits, no parity, no hardware control".
4. After a reset, an initial menu appears in the console.

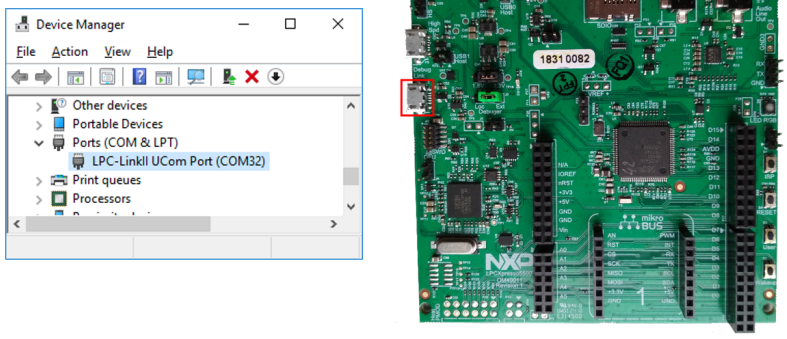

Figure 10. Virtual serial COM port and board

5.2 Application

When the application is running, a menu appears in the terminal window. On the screen, the PUF status and the PUF allowed operation are always displayed.

The PUF peripheral is unable to start since the initial state displayed in the console is out of reset and therefore, no commands are allowed. However, the PUF can be started from the boot code, depending on the CMPA key storage area and the "Disable PUF enrollment" and "Disable PUF key code generation" boot options.

Prior to running, the PUF peripheral must be initialized. The SRAM memory is repowered and made ready to be read during initialization.

The PUF status has the following meaning:

- *Busy*: An operation is in progress.
- *Success*: The last operation has been successful.
- *Error*: The PUF is in the error state and no operations can be performed.

At the start of the PUF-allowed operation, all four status bits have the following meaning:

- `Enroll`: The `Enroll` operation is allowed.
- `Start`: The `Start` operation is allowed.
- `SetKey`: The `SetKey` operation is allowed.
- `GetKey`: The `GetKey` operation is allowed.

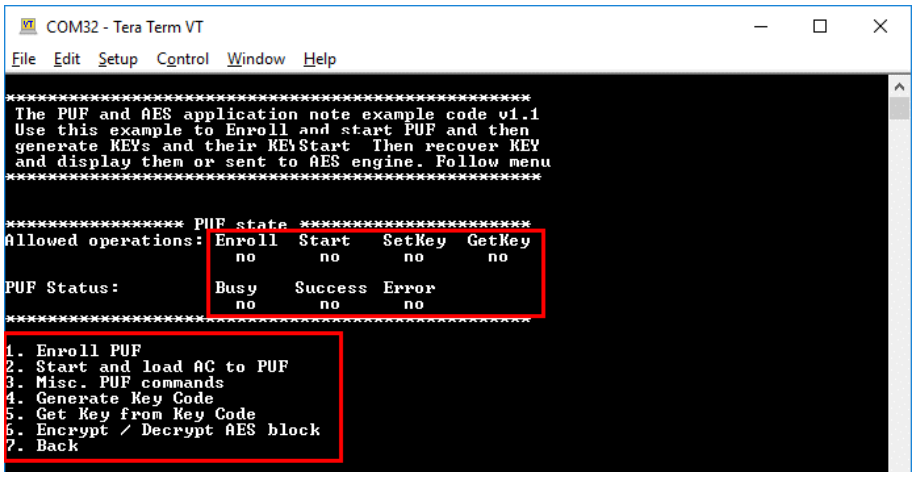

Figure 11. Application main screen

5.3 Enroll PUF

To start using the PUF, the first step is to enroll, as explained in Section 3.3.1. During the enrolling, an AC is created. The enroll operation first powers up the PUF peripheral, initializes it (`Start`), and then calls the `Enroll` command. The AC is created and can be stored to the RAM or flash memory for further use.

Note: *After a successful `Enroll` command, only the `SetKey` command is allowed.*

To call the `Enroll` command again, the PUF peripheral must be power-cycled and initialized again. To enable the `GetKey` function, the PUF peripheral must be power-cycled and started again. The AC can be stored into the RAM or flash memory for further usage.

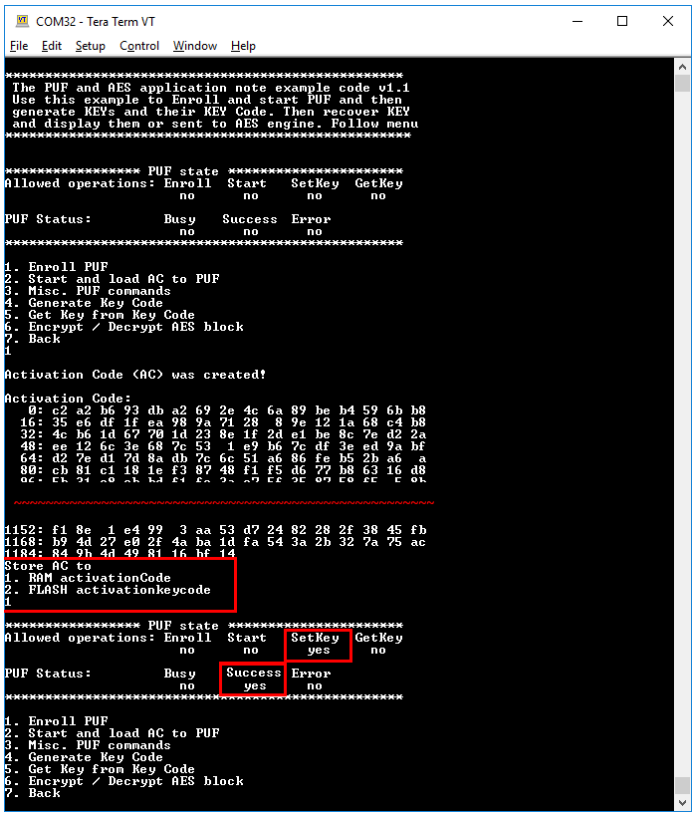

Figure 12. Enroll operation

5.4 Start and load AC to PUF command

The `Start and load AC to PUF` command starts the PUF and loads one of the stored activation codes. The AC must match the key code generated by the same AC. Otherwise, the `GetKey` command fails. The AC can be loaded from the RAM or flash memory that is stored previously or from the CMPA position (created and stored by blhost).

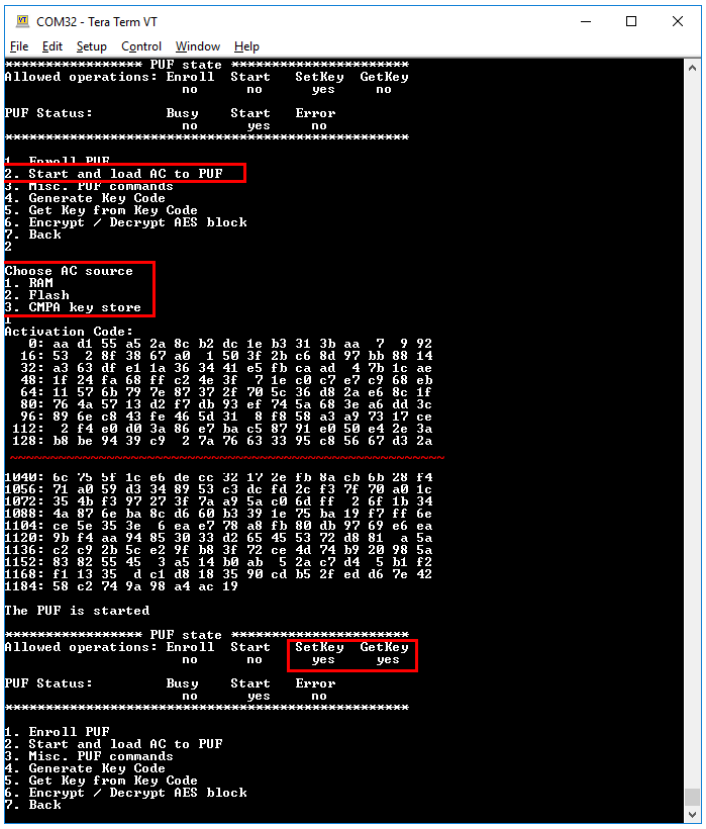

Figure 13. Start and load AC to PUF command

5.5 Generate key code command

To generate the user key, perform the following steps:

1. The `Generate key code` command enables you to create a user key or an intrinsic key as follows:
   - *User key*: A key is entered to the PUF and the key code is generated out of this key.
   - *Intrinsic key*: A key is randomly generated inside the PUF and you get only the key code. If the intrinsic code has index 0, the key is permanently concealed and the CPU cannot read it since the PUF never sends it to the AHB bus.
2. The next step is to set the key index in the interval from 0 to 15. The key retrieved from the key code by the `GetKey` command is sent to the internal hardware bus when the key index is 0. A key index of 1 (and higher) sends the retrieved key to the AHB and the CPU can read it.
3. Enter the key size. The PUF supports key sizes from 64 bits to 4096 bits. However, the PRINCE engine supports only 128-bit keys and AES 128/196/256-bit keys.
4. Enter the user key size. The terminal supports only ASCII codes for simplicity reasons.

The entered parameters are sent to the PUF peripheral. The key code is generated and printed.

The generated key code can be stored in one of the four available places:

- Two key codes in the RAM.
- Two key codes in the flash memory for further use.

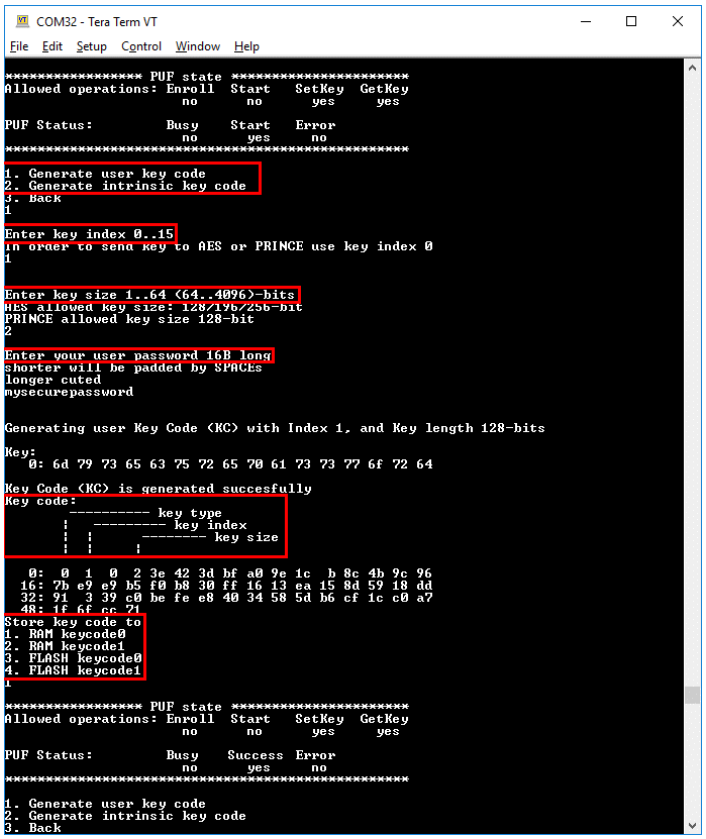

Figure 14. SetKey operation

5.6 GetKey command

Use the `GetKey` command to retrieve a key back from a key code. If a key code index is higher than 0, the key is reconstructed and printed.

If a key code index equals 0, the key is sent to a secret internal hardware bus. This bus is connected to the AES and PRINCE engines. Enter a keyslot value of 1 to send the key to the AES engine. Values from 2 to 4 send the key to PRINCE1, PRINCE2, or PRINCE3. If the command is successful, the key is temporarily stored in the AES engine and can be used to encrypt/decrypt a 16-byte block of data.

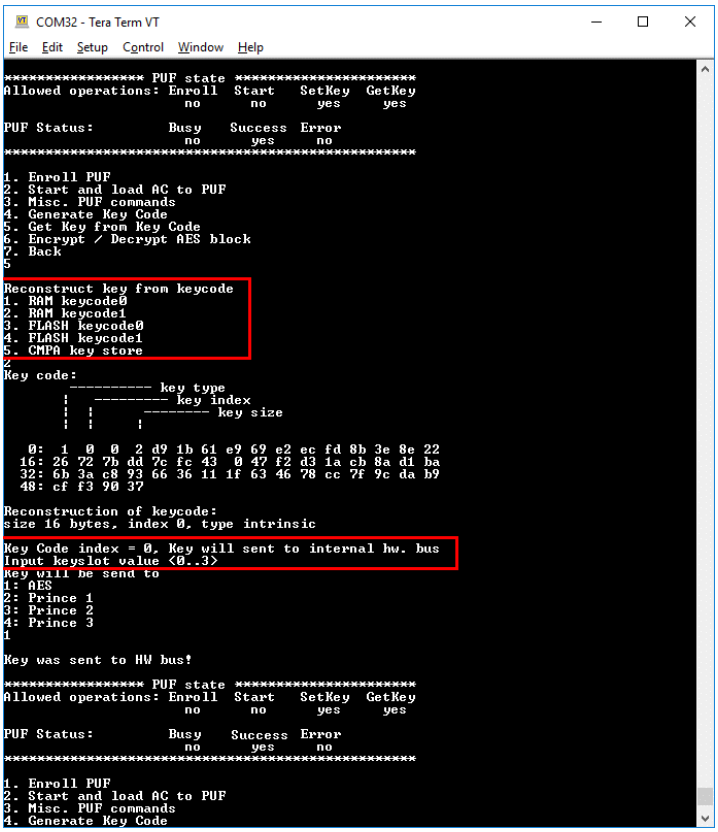

Figure 15. GetKey operation - internal hardware bus

Key codes with a key index higher than 0 have their key printed to the console.

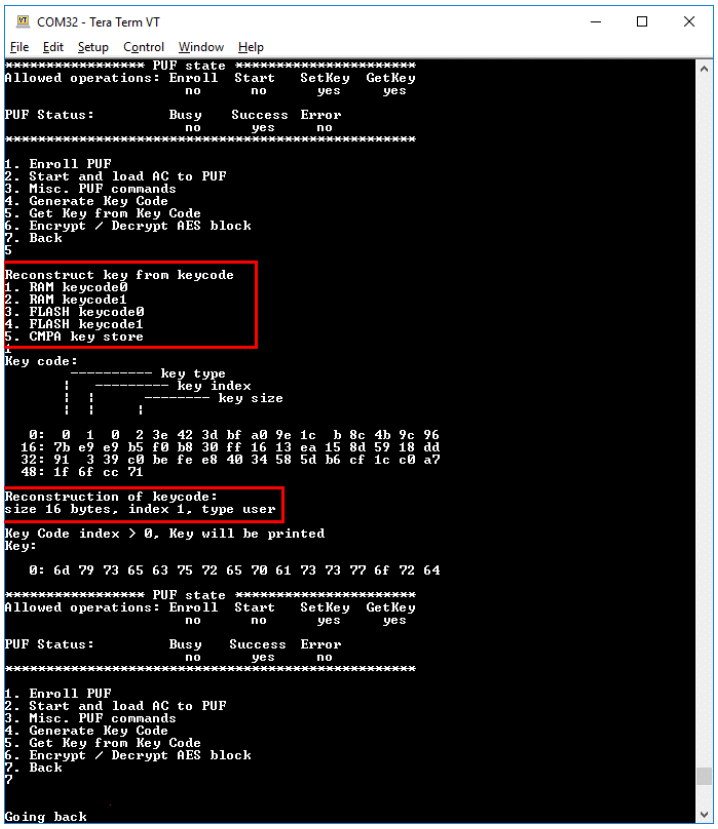

Figure 16. GetKey command - key is printed

5.7 Using a key to encrypt a block of data

This command enables you to encrypt/decrypt a 16-byte block of data using a key reconstructed by the PUF.

You can choose either between a secret key or a user key. Secret key is sent to the AES through an internal secret hardware bus using the `GetKey` command whereas the PUF generates the user key from one of the stored key codes.

Enter the plain text in ASCII. This block is then encrypted/decrypted using the provided key. AES uses the ECB mode.

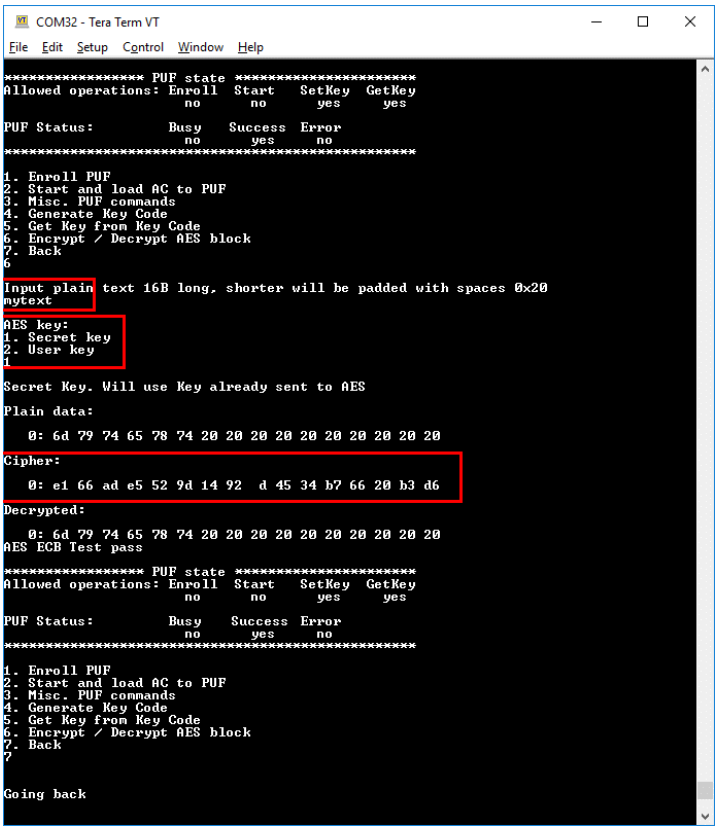

Figure 17. Encrypt/decrypt menu

5.8 Miscellaneous menu

The miscellaneous menu has the following options:

- Init PUF
- Stop PUF
- Zeroize PUF
- Disable Enroll PUF
- Disable Key Generation

The "Disable Enroll PUF" and "Disable Key Generation" settings can be cleared only by power-cycling the MCU.

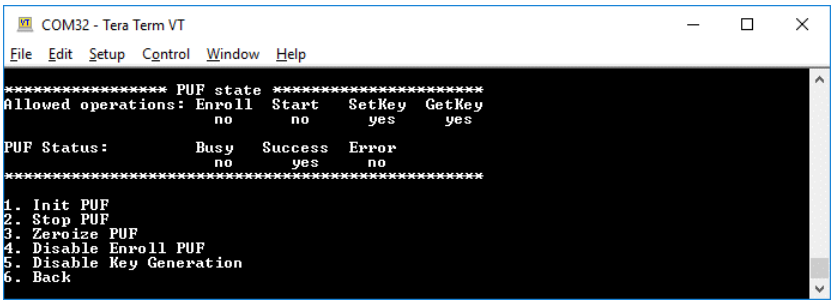

Figure 18. Miscellaneous menu

6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

7 Revision history

Table 3 summarizes the revisions done to this document.

Table 3. Revision history

Revision history	Release date	Description
2	22 September 2023	Section 1 is updated.
1	31 August 2023	• Multiple editorial changes • Figures updated to svg format • Various codeblocks incorporated • Spelling and grammar improvements for the entire document • Figure 11 and Figure 12 updated • Added Section 6
0	20 February 2019	Initial public release

8 Legal information

8.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

8.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

For more information, please visit: http://www.nxp.com

Date of release: 22 September 2023
Document identifier: AN12324