AN14361

Generating Digital Signature using ELS ECSIGN Command with RTF enabled

Rev. 2.0 — 13 July 2026 Application note

Document information

Information	Content
Keywords	AN14361, MCX, MCX N series, digital signature, ECSIGN, RTF, ELS
Abstract	This application note focuses mostly on the ECSIGN command and creates a demo to use it with RTF enabled.

1 Introduction

The MCX N series is a family of Arm Cortex-M33-based microcontrollers for embedded applications. It features advanced serial peripherals, CoolFlux BSP32, a PowerQuad DSP co-processor, and enhanced security features supporting a wide variety of applications.

This family of microcontrollers supports different security features including the ELS and PKC modules. These modules offer support for secure boot, AES, ECDSA, UUID, dynamic encrypt and decrypt, debug authentication, and DICE.

Note: *This application note includes specific examples for the MCX N94; however, the information presented in this application note is also applicable to the MCX N54, N53, N52, N24, and N23 devices.*

1.1 EdgeLock Secure Subsystem (ELS)

As the usage of IoT devices continues to surge, the risk of these devices being vulnerable to various exploits is also increasing. Therefore, it becomes crucial for users to have devices where their data is secure, hidden, and unexploitable. This is where ELS comes in as a solution.

The ELS, also known as EdgeLock Secure Enclave - Core Profile (ELE-CP), provides a secure vault for device credentials and platform integrity keys. It provides the highest level of isolation for executing sensitive security functions, including cryptographic operations or SoC integrity protection functions.

The MCX N series has an ELS module. This security subsystem provides a strong key isolation from the SoC and supports a wide range of cryptographic algorithms. These algorithms include AES, HMAC, HMAC based HKDF, CMAC, and so on. ELS is the main building block of a SoC immutable root of trust. It is used as a part of the trust anchor during secure boot, secure debug access, life-cycle management, and trust provisioning. AES operations of ELS are configured to provide resistance against side-channel attacks. ECC P-256 operations provide protection against side-channel attacks and low-cost fault attacks, such as non-localized EMFI, power supply, and clock glitch attacks. For more detailed information on ELS, refer to the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM)[1].

1.1.1 ELS commands

The primary user interface to ELS is via the ELS commands. These commands are listed in Table 1. In the table, only a brief description is provided. For more information about the commands and their usage, refer to the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).

Table 1. List of ELS commands

Command	Command ID (decimal)	Description
CIPHER	0	This command runs a block cipher within one of the cipher modes, ECB, CBC, CTR, as described in NIST SP 800-38a "BLOCK CIPHER MODES".
AUTH_CIPHER	1	This command runs an authenticated block cipher as described in NIST SP 800-38d_topic_is_GCM_BLOCK_CIPHER_MODE.
ECSIGN	4	This command runs an ECDSA P-256 sign operation.
ECVFY	5	This command runs an ECDSA P-256 signature verification operation.
KEYGEN	8	This command runs an ECDSA P-256 diffie-hellman key exchange calculation to create a shared secret.
KEYIN	9	This command creates an asymmetric key pair.

1 To access this document, contact your local NXP field applications engineer (FAE) or sales representative.

Table 1. List of ELS commands*...continued*

Command	Command ID (decimal)	Description
KEYOUT	10	This command “wraps” (encrypts) an ELS internal key using the RFC3394 algorithm and stores the encrypted key in system memory.
KDELETE	11	This command deletes a key from the ELS internal keystore.
CKDF	16	This command derives (creates) a new key from an ELS master key and stores the new key in the ELS internal keystore using the NIST SP 800-108 algorithm.
HKDF	17	This command derives (creates) a new key from an ELS master key and stores the new key in the ELS internal keystore. HKDF uses either the RFC5869 algorithm or the hash-based option in the NIST SP 800-56C algorithm.
TLS_INIT	18	This command creates session keys as defined by the TLS 1.2 standard.
HASH	20	This command hashes a message using the NIST FIPS 180-4 (SHA2) standard.
HMAC	21	This command creates an authenticated message hash as defined by NIST FIPS 198-1.
CMAC	22	This command creates an authenticated message hash as defined by NIST SP 800-38b.
RND_REQ	24	This command creates either a block of pseudo random data as defined by NIST SP 800-90Ar1 or a block of raw random data that is output by the ELS TRNG.
DRBG_TEST	25	This command supports FIPS CAVP testing. It supports three modes.
DTRNG_CFG_LOAD	28	This command loads a prepared TRNG configuration to configure the ELS internal TRNG properly.

This application note focuses mostly on the ECSIGN command and creates a demo to use it with Run Time Fingerprint (RTF) enabled. The MCX N series SDK contains various ELS command example projects including the ECSIGN usage but not with RTF enabled. More information about this specific use case is mentioned in the following sections.

1.2 Run Time Fingerprint (RTF)

RTF is a 256-bit internal register that can be used to support flows that attest system hardware and firmware. It is located in an internal ELS register that is not accessible via the register map. This access restriction prevents attempts to produce a fake copy of RTF, that is, one that has not been created by hashing the system memory. It is initialized to a constant value of 256’h0 whenever there is any ELS reset. An example of such a support flow sequence is as follows:

- ELS can be configured to output the result of the HASH command to RTF. When configured in this way, the current value of RTF is included in the HASH input data. So, the output of each successive hash with RTF depends on the results of all previous hashes with RTF. In this way, HASH with RTF can be used to build a cumulative digest from multiple sections of system memory. This can be used as a representative digest of system firmware. The creator of the firmware must be able to recreate the same digest value independently. A more detailed information on how RTF is updated via the ELS HASH command is as follows:
  1. The input message to HASH is hashed as normal.
  2. Then, the resulting digest is appended to the current value of RTF and hashed again. For more details on the algorithm used to combine RTF with the hashed message, refer to the RTFUPD parameter in HASH parameters in the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).

3. The final RTF value is stored in RTF.
4. The resulting hash (with RTF) is stored to the output buffer provided in the HASH command. The resulting output is as follows: *Output = HASH || RTF*.

- After RTF has been updated from the hashes of the system memory, it can be “added” to a message to be signed (challenge). Then, the result can be signed using the ECSIGN command. For more details on the algorithm used to add RTF to a message before the ECSIGN command signs it, refer to the SIGNRTF parameter in the ECSIGN parameters, in the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM). Because the resulting signature depends on the value of RTF, which in turn depends on the system memory contents, a correct signature provides proof that the system memory contents are as expected and not tampered with.
  Since this flow helps a user verify the system firmware and memory of the device, the above part has been demonstrated in this application note.
- To prove that the attested firmware is running on a trusted hardware, ELS also supports the gathering of a “hardware signature”, the Hardware Attestation Data (HAD) from the system. The actual attestation consists of a challenge-response step. An example is as follows:
  A verifier creates a challenge, which is sent to ELS. ELS concatenates the challenge, the RTF, and the HAD. It then signs the result with a trusted signing key and returns the signature together with the HAD, to the verifier. The verifier has the challenge and HAD and can recreate RTF. So, the verifier can independently recreate the concatenated message and use that to verify the signature returned by ELS.

1.3 ECSIGN command

The ECSIGN command generates an ECC P-256 digital signature. From the ELS keystore, it calculates the signature over a challenge (data) that has been provided by the caller using an ECC signing key. Based on the requirement, the user or one of the pre-generated ELS keys can be used to generate this ECC signing key. This command also has specific byte and bit order requirements.

ECSIGN consists of the following sub-operations:

1. Read the challenge from system memory, and hash it if required. The challenge can be pre-hashed. In this case, hashing is not required.
2. Use the ECC signing key to compute the signature over the hash of the challenge.
3. Write the signature to the system memory.

1.3.1 Use of digital signature in an authentication scenario

The ECSIGN command generates a digital signature. Typically, that is required as part of a “crypto authentication” scenario where ELS is embedded in a “peripheral” that must authenticate itself to a “host”. To do that, the peripheral must hold the private key of an asymmetric key pair, plus a certificate containing the public key of the pair.

The sequence is as follows:

1. The peripheral sends its certificate to the host.
2. The host then verifies and unpacks the certificate to get the public key of the peripheral. In doing so, the host confirms that the public key is authentic.
3. The host generates a random number, “the challenge”, and sends it to the peripheral as a plaintext.
4. Optionally, the peripheral creates a random number “the nonce”, and concatenates it to the challenge.
5. The peripheral signs the challenge using a standard digital signature algorithm. This step requires the private key. The result is the “response”, which the peripheral returns to the host. If the challenge included a nonce, the peripheral must return both the nonce and the response to the host.
6. The host verifies the response using a standard verification algorithm. If the result is ok, the peripheral has proved to be authentic.

The role of ELS in the above sequence is as follows:

- Generate the nonce: ELS does this via the RND_REQ command. This is optional.
- Sign the challenge: ELS does this via the ECSIGN command. ECSIGN performs the operation *{r, s}* = *signature(hash(c))*, where:
  - *{r, s}* is the signature that is output from ECSIGN.
  - *c* is the challenge; a random number that has been received from the host and is located in the memory of the peripheral.
  - hash is a standard SHA256 hash.
  - signature is an ECDSA signature using the NIST P-256 curve.

Table 2. ECSIGN parameters

Parameter	Register	Description
SRC0_ADDR	DMA_SRC0	Start address of the digest/challenge. If ECHASHCHL = 0, it is the start address in the system memory of the digest. The challenge has been pre-hashed, and the digest is the hash of the challenge. If ECHASHCHL = 1, it is the start address in system memory of the unhashed challenge. In this case, the command hashes the challenge, and uses the resulting digest in the ECSIGN operation.
SRC0_LEN	DMA_SRC0_LEN	If ECHASHCHL = 0, the parameter is ignored. A digest has a fixed length. No need to specify the length. If ECHASHCHL = 1, Length in bytes of the unhashed challenge.
RES0_ADDR	DMA_RES0	It is the start address in system memory where the signature is stored.
KIDX0	KIDX0	Index in the ELS keystore of the key used to sign the message.
SIGNRTF	CMDCFG0[1]	0 = RTF is not incorporated in the message to be signed (the challenge). 1 = RTF is incorporated into the message to be signed (the challenge). The following expression is an algorithm, which is used to incorporate RTF: *signature = ECDSA P-256 (signing key, SHA256(RTF || HAD || message_to_be_signed_with_padding_adjusted_for_sizeof_RT F_plus_HAD))* ***Note:*** *If SIGNRTF = 1, then at least one of the URTF or UECSG must be set on the signing key.*
ECHASHCHL	CMDCFG0[0]	Indicates whether the challenge has been pre-hashed. If the challenge has not been hashed, the command hashes the challenge to create a digest and use the digest in the ECSIGN operation. 0 = The challenge is pre-hashed (a digest). The command uses that directly in the ECSIGN operation. 1 = The challenge is not pre-hashed. The command hashes the challenge, and uses the resulting digest in the ECSIGN operation.
REVF	CMDCFG0[4]	0 = The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word little-endian format. 1 = The digest and the signature (but not the challenge, see below) are accessed (read/written) with the assumption that they are in 32-bit word big-endian format. ***Note:*** *REVF is ignored when reading the challenge. The challenge is always read with the assumption that it is in 32-bit word big-endian format.*

Table 3. ECSIGN data structures

Object	Description
Key	A 256-bit keystore key must be used.
Input message: “the challenge”	If ECSIGN is configured to get the challenge from system memory, the user must place the challenge there. Otherwise, no action is required because ELS already holds the challenge internally. If the challenge is to be hashed, its size must be an integer number of SHA-256 blocks. A SHA2-256 block is 512 bits. If the challenge has already been hashed, its size must be 512 bits.
RTF	This is a 256-bit ELS internal register that contains the result of one or more hashes of system memory. When the command switch SIGNRTF = 1, RTF is concatenated with the input message. The HAD and the result are signed. *Concatenation = RTF || HAD || input message*.
HAD	This is a block of system memory that contains the data that represents the hardware “state” of the system. When a command with SIGNRTF = 1 is used, HAD is concatenated with the input message and RTF, and the result is signed. *Concatenation = RTF || HAD || input message*.
Signature	The signature is written to system memory by ECSIGN. Its size is 512 bits. It consists of the concatenation of two 256-bit components: R and S. *Concatenation = R || S*. If R || S is located in system memory, R must be at lower addresses and S must be at higher addresses.

1.3.2 ECSIGN command without RTF enabled

Below is the sequence to use the ECSIGN command without enabling RTF:

1. The user prepares the input message. This input message can either be a plain message or a hash of the message.
   If the input message is a pre-computed hash of the input message, then the "ECHASHCHL" bit must be set as 0.
   Otherwise, if the input message is a plain message, the "ECHASHCHL" bit must be set as 1. So that the ECSIGN command hashes the challenge and uses the resulting digest for the signing operation. Also, in this case, the user must take care of the padding. Standard SHA2-256 padding must be added with the plain message (based on FIPS-180-4 section 5.1).
2. The user sets the command parameters and starts ECSIGN. Only the RTF related parameter is mentioned (SIGNRTF = 0). Other parameters must also be prepared as required. For more information, refer to Table 2 or the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).
3. ECSIGN completes. The signature is output to the system memory.

An SDK example project is already available (within the MCX N SDK package) for this use case. For more information on ECSIGN parameters, refer to the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).

1.3.3 ECSIGN command with RTF enabled

Below is the sequence to use the ECSIGN command with RTF enabled:

1. The user prepares the input message. When the RTF must be used, the input message must be unhashed and padded.
   Padding (standard SHA256 padding, based on FIPS-180-4 section 5.1) must be done in the following way: The message that is signed is the concatenation of RTF, HAD, and the input message. The ELS internally concatenates the RTF and HAD but the user must provide padding for them.

Concatenation = RTF || HAD || Input Message
Therefore, the input data to be provided by the user must have the input message. The standard SHA-256 padding considers the RTF and HAD. To demonstrate an example, take the input message as follows:

```
Input_message[] = {0x11111111, 0x22222222, 0x33333333};
```

Here, *Input_message* is 12 bytes. Also, the RTF and HAD are 32 bytes and 48 bytes, respectively (for MCX N series devices). Therefore, the actual size of the buffer ECSIGN command is:
Total size = 32 (RTF) + 48 (HAD) + 12 (Input message) + 36 (adding padding bytes such that the whole buffer is multiple of 64) bytes
Now, because ELS internally adds RTF and HAD, the actual buffer (to be passed by the user) only includes the input message and the padding bytes. The final message that we pass to the ECSIGN command must be as follows:

```
Final_message_padding_for_rtf_plus_HAD_and_whole_msg [] =
{0x11111111, 0x22222222, 0x33333333, 0x00000080, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0xE0020000};
```

The fourth word includes 0x80 bytes to state that the data was present until that point. The last word includes the size of the whole data in big-endian unsigned integer format. Both these changes are part of the standard SHA256 padding:
32-bytes RTF + 48-bytes HAD + 12-bytes Input message = 92-bytes = 736 bits (0x2E0)

2. The user sets the command parameters and starts ECSIGN. Only the RTF related parameter is mentioned (SIGNRTF = 1). Other parameters must also be prepared as required. For more information, refer to Table 2 or the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).
3. ECSIGN completes. The signature is output to system memory.
   To verify the generated signature (also done in the attached example project), the user must prepare the RTF and HAD, and then add them in the buffer such that:
   Message_for_verification = RTF || HAD || Input Message || Padding
   Pass this to the ECVFY command. ECVFY command requires other parameters as well. For more details, refer to the *MCX N Security Reference Manual* (document MCXNP184M150F70SRM).
   Here again, standard SHA-256 padding is used.

2 Setup and SDK example

This section explains the hardware and software setup to run the attached SDK example project.

2.1 Hardware setup

For the hardware setup, the FRDM-MCXN947 development board is used, which is connected to the host computer through the J17 USB Type-C connector. The SWD debugger on this board uses the MCU-LINK VCOM output. It acts as a serial to USB bridge to the host computer and providing the SWD debug interface. The MCU debugger is programmed and configured as a CMSIS DAP debugger.

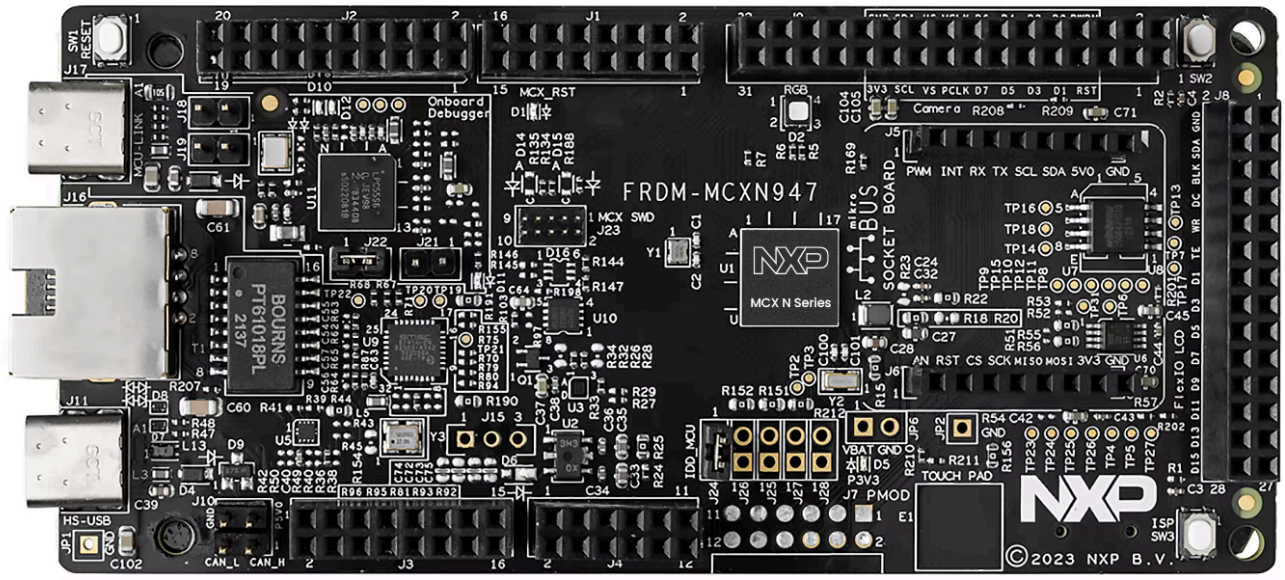

Figure 1. FRDM-MCXN947 development board

2.2 Software setup

For the software setup, MCUXpresso IDE v11.9.0 is used to build and write the image, described later in the application note. The software is available at MCUXpresso-IDE.

An SDK for the MCXN947 is required for building and debugging the code. The application note is attached with the example project. SDK_2.14.0 (available for download at MCUXpresso SDK Builder) is used for the below example project. The output can be displayed on any terminal application such as Tera Term. While using a terminal application, the below settings must be used:

- 115200 baud rate
- No parity
- 8 data bits
- 1 stop bits

2.3 SDK example project

An example project is attached to this document to display this feature. To import the example project (archived folder), perform the following steps:

1. Open the MCUXpresso tool.
2. From the Quickstart panel, select “Import Project(s) from file system” > “Browse”.
3. Then, browse the zip folder and select “Finish”.

The project is now available under the Project Explorer window. To build the project, click the hammer icon on the top panel, as shown in Figure 2.

To flash the project into the MCU and simultaneously start a debugging session, select the “Debug” option from the “MCUXpresso IDE – Quickstart Panel”. Alternatively, click the blue bug button from the top panel, as shown in Figure 2.

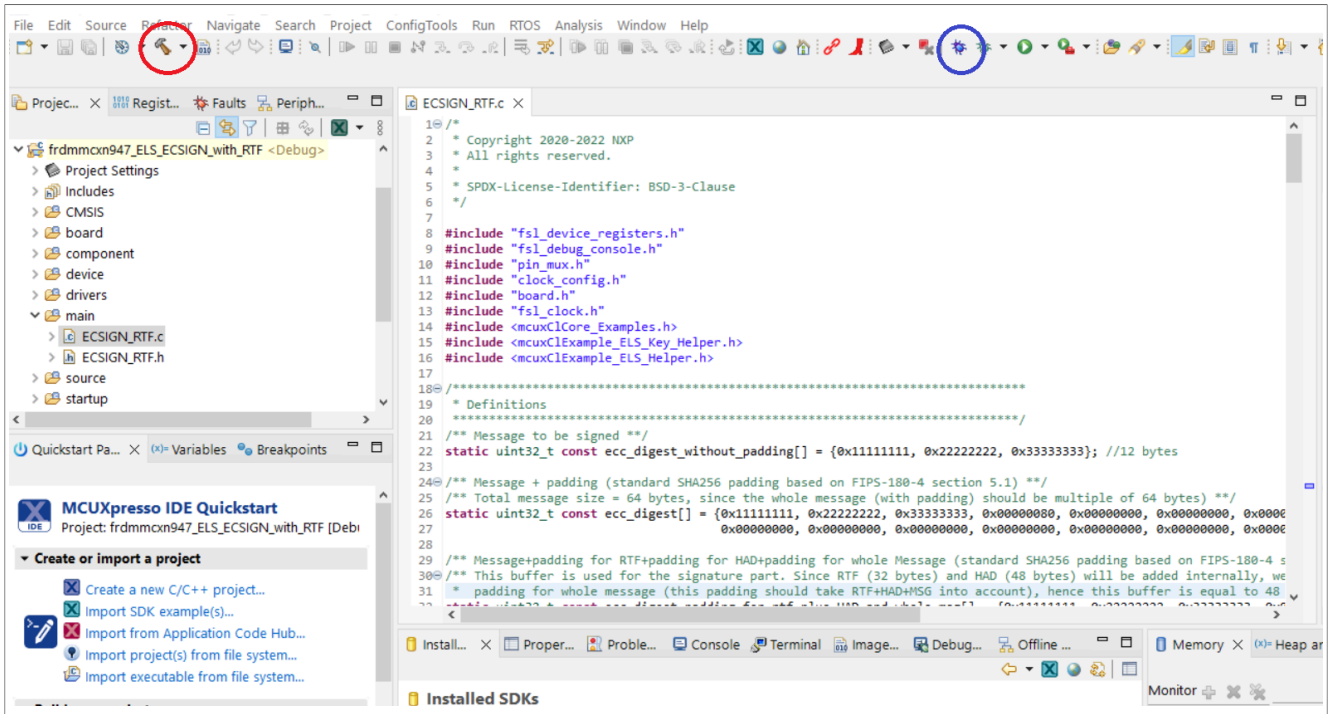

Figure 2. Hammer icon (red circle) and bug icon (blue circle)

2.4 Output

The output is displayed in the Tera Term application using the UART COM41 port. After the successful build and flashing of the code into the MCU, the user can see the output as shown in Figure 3. Figure 3 shows that the RTF and signing key were generated and then the data is successfully signed with RTF enabled and verified.

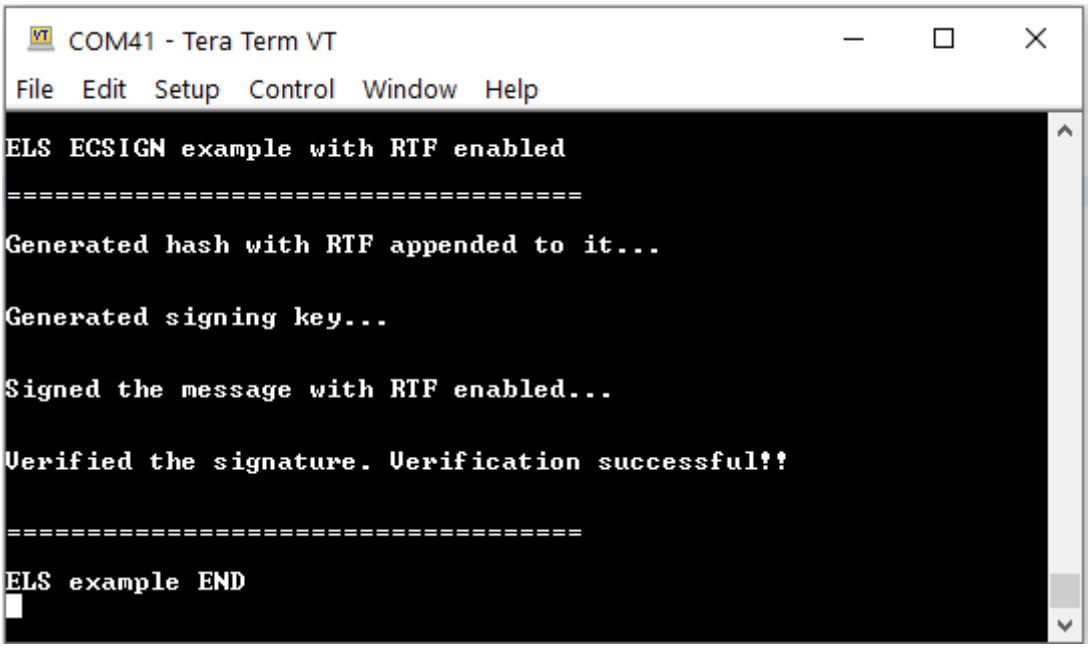

Figure 3. Output of the demo

3 Acronyms

Table 4 describes acronyms used in this document.

Table 4. Acronyms

Acronym	Description
AES	Advanced Encryption Standard
CMAC	Cipher-Based Message Authentication Code
DICE	Device Identifier Composition Engine
ECDSA	Elliptic Curve Digital Signature Algorithm
ELE	EdgeLock Secure Enclave
ELE-CP	EdgeLock Secure Enclave - Core Profile
ELS	EdgeLock Secure Subsystem
EMFI	Electromagnetic Fault Injection
HKDF	Extract-and-Expand Key Derivation Function
HMAC	Keyed-Hash Message Authentication Code
PKC	Public Key Cryptography
RTF	Run Time Fingerprint
UUID	Universally Unique Identifier

4 Note about the source code in the document

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

Copyright 2026 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.

5 Revision history

Table 5 summarizes the revisions to this document.

Table 5. Revision history

Document ID	Release date	Description
AN14361 v.2.0	13 July 2026	Added a note in Section 1 "Introduction"
AN14361 v.1.0	29 July 2024	Initial public release

Legal information

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.

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 https://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 document 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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

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.

Tables

Figures

Contents