AN12662

Binding a host device to EdgeLock SE05x

Rev. 1.1 — 7 December 2020 Application note

Document information

Information	Content
Keywords	EdgeLock SE05x, binding, secure, boot, PUF, secure channel
Abstract	This application note introduces the concept of binding as a way to establish a secure, encrypted and authenticated channel between EdgeLock SE05x and a specific MCU, and clearly highlights its importance for the security of an IoT device.

Revision history

Revision history

Revision number	Date	Description
1.0	2020-07-06	First version
1.1	2020-12-07	Updated to latest template and fixed broken URLs

1 Introduction

Internet of Things (IoT) devices have reached a widespread use in many different applications, including industrial and automotive equipment requiring high reliability and robustness. Such IoT devices may have access to sensitive end-user data, critical sensor information and intellectual property in the form of software implementations and algorithms. Specifically in industrial applications, operation continuity and correct functioning is essential. For these reasons, hardening IoT devices against logical and physical attacks becomes more and more important. Applying strong cryptography and security best-practices enables a high level of security for important assets.

A possible solution to the challenges listed above is the integration of a dedicated Secure Element (SE) such as EdgeLock SE05x into the IoT device. The SE protects mission-critical cryptographic keys and provides cryptographic services to the device. EdgeLock SE05x is security certified to a level of CC EAL 6+ and provides security against physical and logical attacks aimed, for example, at extracting security keys.

EdgeLock SE05x provides a root of trust and a trusted identity to the device and serves as a secure key vault. Additionally, EdgeLock SE05x provides manufacturers the option to bind the MCU of the IoT device to the secure element, so that security services offered by EdgeLock SE05x can only be used by that particular MCU.

The binding process can be implemented at different stages of the product manufacturing and can be adapted to support incremental security levels depending on the IoT device security requirements and the available MCU security features.

2 Binding EdgeLock SE05x to a host MCU

Binding is a process to establish a pairing between the IoT device host MCU and EdgeLock SE05x, so that the MCU is only able to use the services offered by the paired EdgeLock SE05x and EdgeLock SE05x is only able to provide services to the paired MCU. A mutually authenticated, encrypted channel will ensure that both parties are indeed communicating with the intended recipients and that local communication is protected against local attacks, including man-in-the-middle attacks aimed at intercepting the communication between the MCU and the SE and physical tampering attacks aimed at replacing the MCU or the SE.

EdgeLock SE05x natively supports Global Platform Secure Channel Protocol 03 (SCP03) for this purpose.

2.1 Secure Channel Protocol 03

The SCP03 protocol is defined in Global Platform Secure Channel Protocol '03' - Amendment D v1.2 specification. The protocol can be used to establish a secure, encrypted and authenticated channel between an off-card entity or host (e.g. the MCU) and a card entity (e.g. EdgeLock SE05x). The establishment of an SCP03 channel requires three static 128-bit AES keys shared between the two communicating parties: Key-ENC, Key-MAC and Key-DEK.

Key-ENC and Key-MAC keys are used during the SCP03 channel establishment to generate, respectively, a session key for encryption / decryption of exchanged data (S-ENC) and a session key to generate / verify integrity of exchanged data (S-MAC). The Key-DEK key is used to encrypt the new SCP03 keys in case they get updated.

SCP03 provides three levels of security to communicating parties:

- **Mutual authentication:** this is achieved through the process of initiating a secure channel and ensures that both parties are communicating with an authenticated entity. The mutual authentication process produces as a result session keys that are then used to encrypt and verify the integrity of the communication;
- **Data integrity:** the message integrity is checked by comparing the MAC received from the host with the card internally generated MAC. The S-MAC key generated after the mutual authentication is used to check message integrity;
- **Data confidentiality:** exchanged APDUs are encrypted using the S-ENC key generated after the mutual authentication process. This prevents unauthorized parties from accessing data that is being transmitted.

The SCP03 mutual authentication process is depicted in Figure 1:

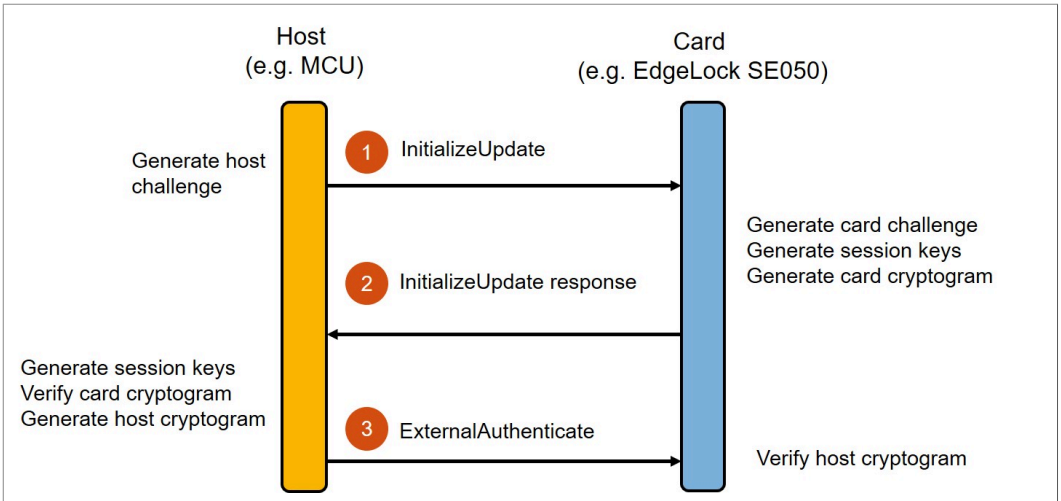

Figure 1. SCP03 mutual authentication process

1. The mutual authentication is always initialized by the host, e.g. the MCU, using the *InitializeUpdate* APDU command. The *InitializeUpdate* command contains random data (host challenge) previously generated by the host;
2. The card entity, e.g. a secure element such as EdgeLock SE05x, receives the host challenge and generates its own card challenge. The host challenge, the card challenge and the static Key-ENC and Key-MAC keys are used to derive the S-ENC and S-MAC session keys. An authentication card cryptogram is also generated using the session keys. The card challenge and the card cryptogram are communicated to the host in the *InitializeUpdate* response;
3. The host now has the same data the card has (host challenge, card challenge, Key-ENC and Key-MAC) and can therefore generate the same session keys and verify the card cryptogram. The host has now authenticated the card. Finally, the host generates the host cryptogram using the session keys and provides it to the card using the *ExternalAuthenticate* APDU command. The card verifies the host cryptogram. If the verification succeeds, the card has authenticated the host and the mutual authentication process is completed.

For more information regarding SCP03 mutual authentication and the involved Global Platform commands, you can refer to GlobalPlatform Card Specification V2.3.1 and Global Platform Secure Channel Protocol '03' - Amendment D v1.2.

The SCP03 protocol also defines a process to create new SCP03 key-sets and update existing key-sets. The process is depicted in Figure 2:

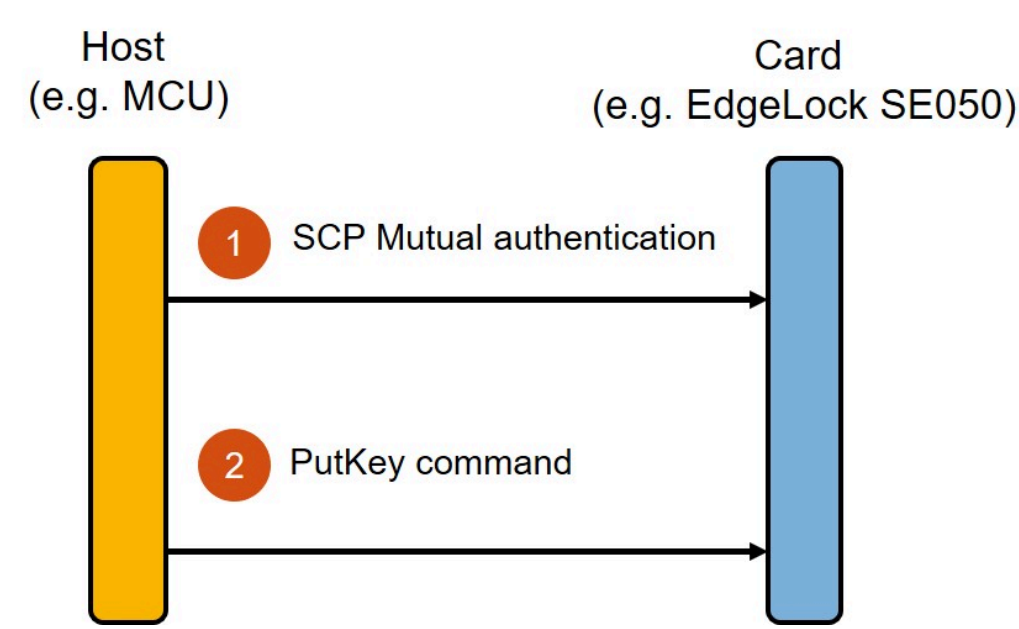

Figure 2. Create or update an SCP03 key-set

1. The host must first authenticate to the card entity using the process described in Figure 1;
2. The host must then use the *PutKey* APDU command to send the SCP03 key-set to the card entity. The *PutKey* command contains in the data field the SCP03 key-set encrypted using the static Key-DEK key. The *PutKey* command also includes the *Key Version* and *Key Identifier* parameters. The *Key Version* parameter identifies the SCP03 key-set that is being updated, while the *Key Identifier* is a unique ID of the key that is being created or updated. The SCP03 keys can be injected all at once using a single *PutKey* command or one by one by issuing multiple *PutKey* commands.

For detailed information regarding the usage of the *PutKey* command, you can refer to GlobalPlatform Card Specification V2.3.1 (Section 11.8).

2.2 Overview of EdgeLock SE05x SCP binding process

EdgeLock SE05x can be bound to the host MCU by injecting in both the MCU and EdgeLock SE05x the same unique SCP AES key-set and by enabling EdgeLock SE05x Platform SCP feature.

EdgeLock SE05x Platform SCP feature allows the MCU to protect the communication between a host processor and the secure element using an SCP03 channel established at the platform level. This secure channel, including the key management operations required to update the default keys, is handled by EdgeLock SE05x using standardized GlobalPlatform commands.

If Platform SCP is made mandatory in EdgeLock SE05x, the MCU would first be required to establish a mutually authenticated SCP session with the secure element before any other operation can be performed. All subsequent APDUs must be sent encrypted and MACed using the SCP session keys.

The binding process between the host MCU and EdgeLock SE05x using Platform SCP consists of a one-time process that can be executed before device shipment or when the device boots up for the first time as shown in Figure 3:

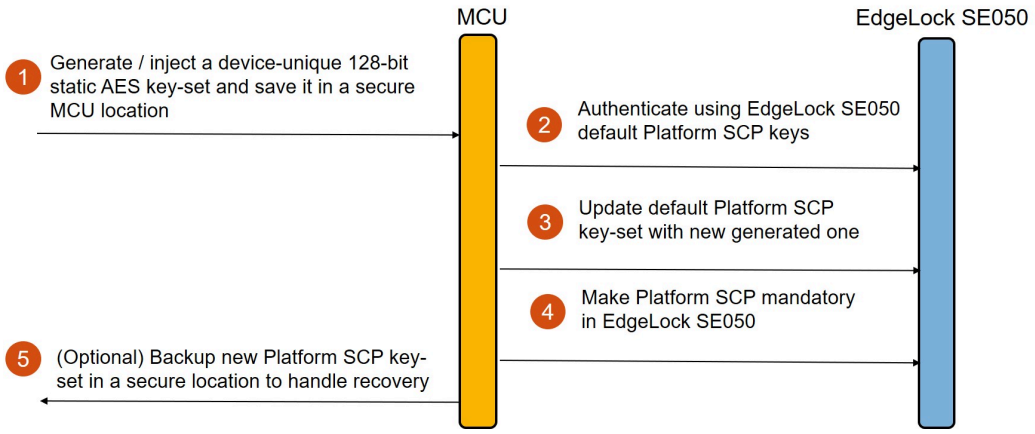

Figure 3. EdgeLock SE05x initial binding process

1. Generate a device-unique 128-bit AES key-set for the MCU when the device first boots up or, for improved security, in the OEMs facilities when the device is manufactured. Alternatively, the key-set can also be remotely provisioned in the IoT device. These keys, from now on referred to as binding keys, must be stored as securely as possible in the MCU;
2. Initialize an SCP03 session as described in Section 2.1 using the default EdgeLock SE05x Platform SCP AES key-set defined here;
3. Update the default Platform SCP key-set with the new AES key-set generated in step 1 using the *PutKey* APDU command as described in Section 2.1;
4. Configure EdgeLock SE05x to require mandatory Platform SCP authentication. This can be achieved by using the EdgeLock SE05x *SetPlatformSCPRequest* APDU to set the *SCP_REQUIRED* flag. From now on, APDUs sent to the EdgeLock SE05xapplet will require full encryption and MAC using Platform SCP03 session keys. Plain APDUs will be rejected by the EdgeLock SE05x applet.
5. Optionally, a backup of the new binding keys can be performed in order to handle the recovery of the device at a later time. If binding keys are generated the first time the device boots up, they can be sent to a secure backend using secure communication protocols (e.g. TLS). If keys have been pre-injected at manufacturing, they should be stored in a secure location in the facilities of the OEM or chip manufacturer.

After these steps have been performed, the MCU is required to establish a Platform SCP channel every time it needs to use EdgeLock SE05x services as shown in Figure 4.

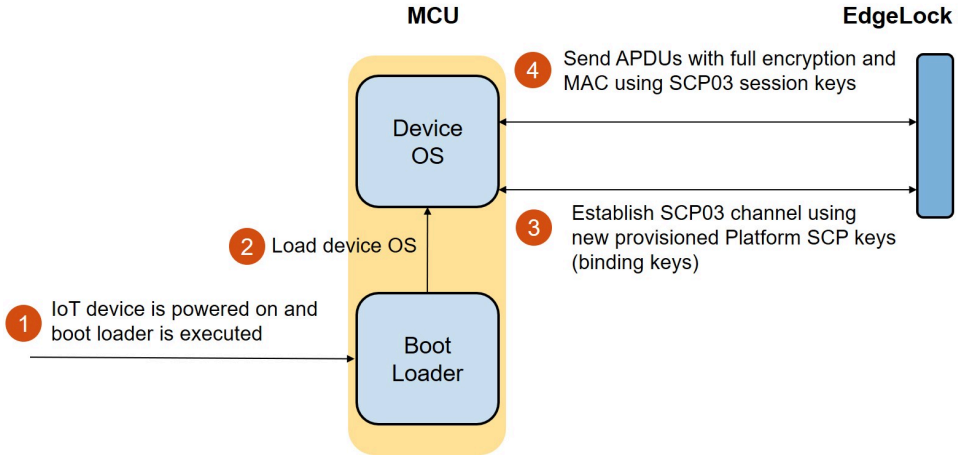

Figure 4. EdgeLock SE05x SCP channel establishment from device OS

Note: *For improved security, the establishment of the SCP channel can be performed by a trusted application, for example by the MCU bootloader. An example configuration is described in* Section 4.

2.3 Ease binding configuration with EdgeLock SE05x Plug & Trust Middleware

The EdgeLock SE05x Plug & Trust Middleware eases the integration effort required to implement the binding process described in Section 2.2. This section gives an overview of some general binding examples that are provided as part of the EdgeLock SE05x Plug & Trust Middleware package and that can be used as a starting point to implement more specific use cases. Examples targeting specific MCU features will be covered in the relevant subsections of Section 3.

The EdgeLock SE05x Plug & Trust Middleware provides the following general examples:

- **SE05X Rotate PlatformSCP Keys Demo**: Showcases authentication with default Platform SCP keys and the rotation (update) of those keys with user defined keys. The example documentation is available in the EdgeLock SE05x Plug & Trust Middleware documentation (`simw-top/doc/demos/se05x/se05x_RotatePlatformSCP03Keys/Readme.html`). The example source code is available at `/simw-top/demos/se05x/se05x_RotatePlatformSCP03Keys`.
- **SE05X Mandate SCP example**: Showcases how to make Platform SCP authentication mandatory in EdgeLock SE05x. The example documentation is available in the EdgeLock SE05x Plug & Trust Middleware documentation (`/simw-top/doc/demos/se05x/se05x_MandatePlatformSCP/Readme.html`). The example source code is available at `/simw-top/demos/se05x/se05x_MandatePlatformSCP`.

Figure 5 shows the binding steps covered by the examples:

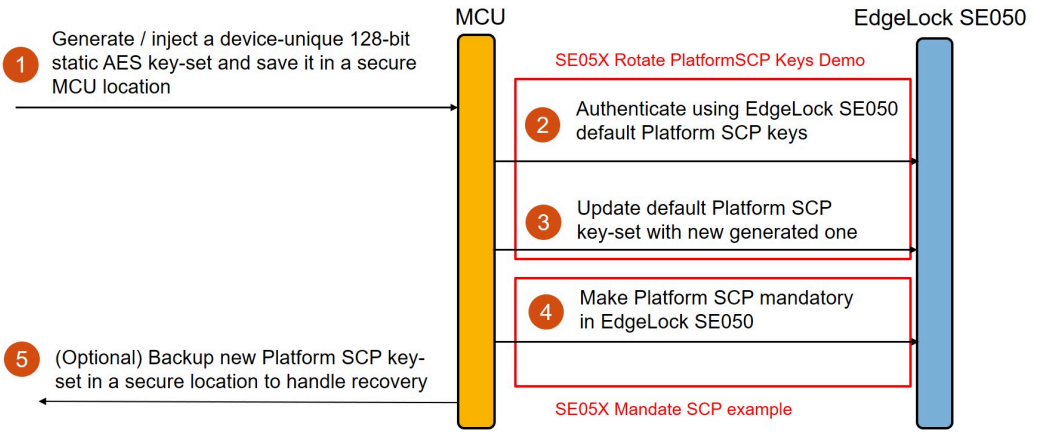

Figure 5. EdgeLock SE05x Plug & Trust Middleware examples applying to initial binding process

The *SE05X Rotate PlatformSCP Keys Demo* takes as input the current EdgeLock SE05x Platform SCP keys (e.g. the default keys of the EdgeLock SE05x sample) and changes them with new keys defined by the user. You can define the current keys by changing the value of the following variables:

- *OLD_KEY_ENC[]*;
- *OLD_KEY_MAC[]*;
- *OLD_KEY_DEK[]*

The new keys should be specified in:

- *EX_SSS_AUTH_NEW_ENC_KEY*;
- *EX_SSS_AUTH_NEW_MAC_KEY*;
- *EX_SSS_AUTH_NEW_DEK_KEY*

The *tp_PlatformKeys()* function opens a Platform SCP session with EdgeLock SE05x using the current keys and rotate them with the new keys defined by the user. Internally, the function builds the Global Platform *PutKey* APDU command and sends it to EdgeLock SE05x. In the example, the *tp_PlatformKeys()* function is run twice, the second time to revert to the old keys. You can comment out the second call to make the key rotation permanent. The new rotated keys are saved and are automatically used by the middleware in subsequent Platform SCP authentications. More information on how the middleware uses custom Platform SCP keys can be found in the EdgeLock SE05x Plug & Trust Middleware documentation (`simw-top/doc/appendix/platfscp.html`).

The *SE05X Mandate SCP example* uses the *SetPlatformSCPRequest* APDU to mandate the use of Platform SCP. The example opens a session with the EdgeLock SE05x secure object *kSE05x_AppletResID_PLATFORM_SCP* and sets its value to *kSE05x_PlatformSCPRequest_REQUIRED* (0x01). This is done through the *Se05x_API_SetPlatformSCPRequest()* API call. After this example is executed, EdgeLock SE05x will require Platform SCP authentication and full encryption and MAC of all APDUs.

Both examples need to be compiled with the CMake flag values shown in Figure 6. If the middleware is compiled with these flags, Platform SCP authentication is always performed by the middleware and all APDUs sent to EdgeLock SE05x will be encrypted and MACed using SCP session keys.

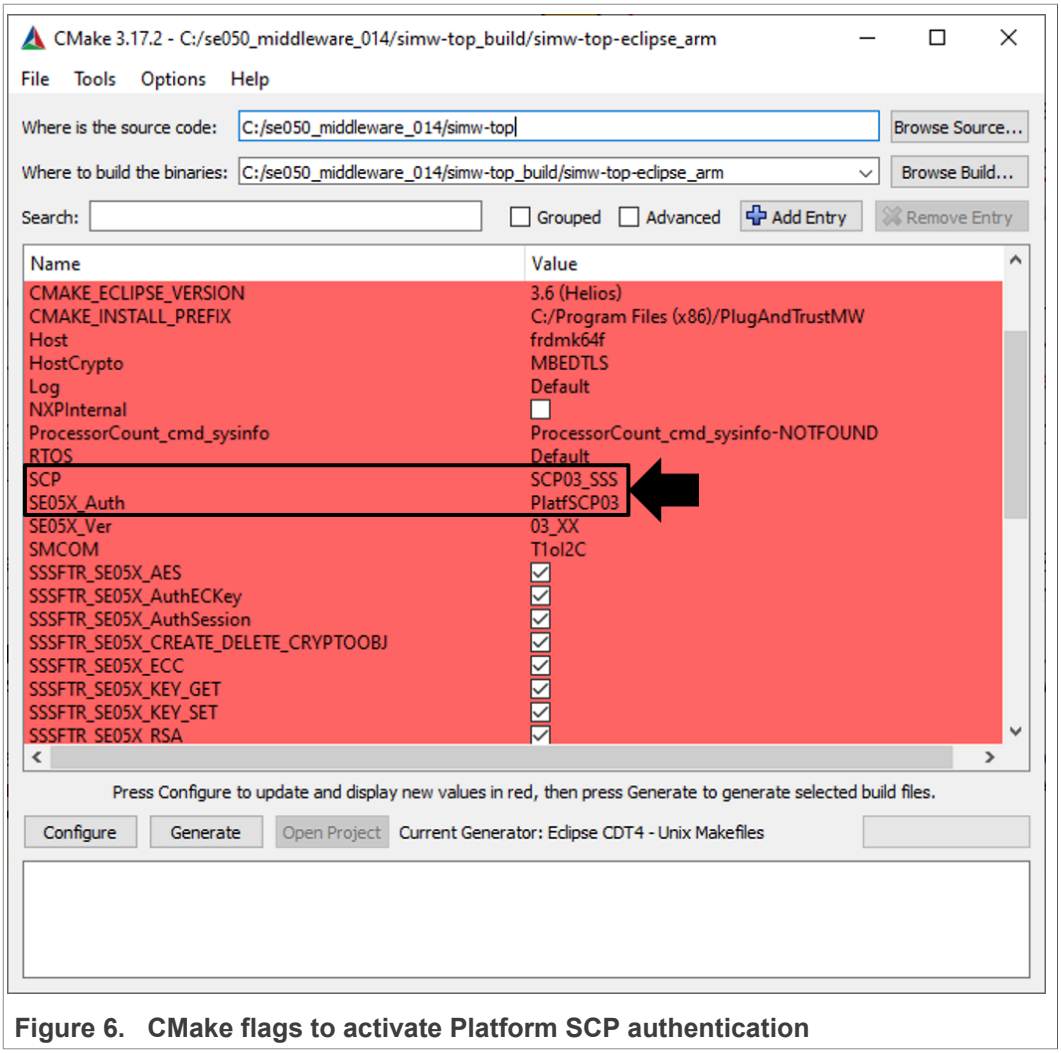

Figure 6. CMake flags to activate Platform SCP authentication

3 EdgeLock SE05x binding configurations

The overall security of the binding process described in Section 2.2 depends on two main factors:

- **The ability of the MCU to protect the shared binding keys**: MCUs provide different security mechanisms for this purpose, from simpler ones such as trust zones to more advanced ones such as Physically Unclonable Functions (PUFs). More security features or the combination of more than one security feature makes the extraction of the binding key from the MCU progressively more difficult for an attacker. Regardless of the security mechanisms that are in place, keys stored in MCUs are still vulnerable to a different degree to advanced attacks such as side channel and fault attacks. In the unlikely circumstance of successful extraction of binding keys from the MCU, EdgeLock SE05x still ensures that credentials stored in the SE cannot be cloned or extracted.
- **The environment and the point in time in which the binding key is generated and injected in the MCU and in the SE**: the injection can be done early in the manufacturing process by the chip manufacturer or the OEM or even after the deployment of the IoT device. Early injection prevents malicious parties down the manufacturing or deployment chain to tamper with the device. The injection process should be performed in a controlled and secure environment using secure established processes to obtain the maximum security level.

Taking into consideration these two variables, the next sections will outline some examples of binding configurations, in order of progressive level of security offered. For each configuration we will discuss how to leverage MCU features to improve the overall security of the binding process. The considered binding configurations are depicted in Figure 7.

Note: *the binding configurations described in this section are provided as examples. Security features can be combined differently to match the security requirements of the specific target use case.*

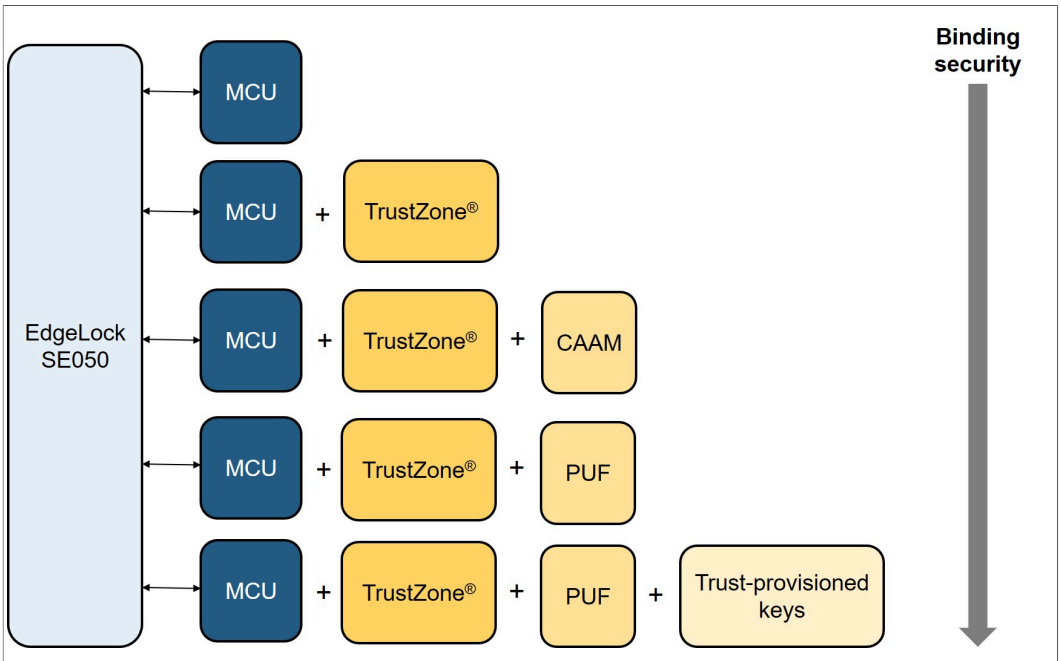

Figure 7. Binding configurations in order of increased security

3.1 Binding for standard MCUs

Most standard MCUs do not provide any security mechanism for protecting sensitive keys from being extracted or manipulated. Most likely, keys would have to be stored in an external flash memory. For this reason, adding an SE such as EdgeLock SE05x improves the overall security of the IoT device. Keys are securely stored in the SE hardware secure memory and never leave the SE boundaries. This feature, coupled with EdgeLock SE05x strong tamper resistance, effectively prevents attackers from accessing the value of stored keys. Moreover, EdgeLock SE05x is pre-provisioned for ease of use with device-unique keys that have been generated and injected in the SE in NXP's secure manufacturing facilities. This allows the user to establish a strong root of trust right from the start that can be leveraged to achieve the most demanding security goals.

Binding a standard MCU to EdgeLock SE05x and establishing a secure SCP channel allows you to improve even more the inherent security introduced by adding an SE. In fact, this security measure helps preventing attackers from accessing EdgeLock SE05x services without proper authorization and from eavesdropping the local communication between the MCU and EdgeLock SE05x. This configuration, together with EdgeLock SE05x tamper resistance, effectively provides increased protection for the IoT device.

In high-end, mission-critical IoT devices, a higher level of protection for binding keys stored in the MCU might be desirable and, in some cases, even mandatory. Adopting MCUs with more advanced security features, as described in the next sections, would make tampering with binding keys stored in the MCU much more difficult for attackers.

3.2 Binding for MCUs with TrustZone®

Nowadays many MCUs come with means to partition the available hardware and software resources into a non-secure world and a controlled, secure world. This security feature allows the user to isolate untrusted code running in the non-secure world and prevent it from directly accessing the resources in the secure world. Only a small subset of selected, trusted applications are executed within the secure world. Standard applications running in the non-secure world can only access services of the secure world through a few, well-defined APIs.

Arm® TrustZone® is a specification of such architecture for Arm-based microprocessors. TrustZone® partitions the available hardware and software resources into a secured Trusted Execution Environment (TEE) and a non-secure Rich Execution Environment (REE). The separation between the secure world and the non-secure world is enforced by hardware level protections.

MCUs implementing TrustZone® typically provide means to securely store keys. This typically takes the form of a write-once memory location accessible only to TEE applications, but different MCUs might offer more advanced mechanisms and varying degrees of protection for keys. The hardware and software isolation provided by TrustZone® makes it more difficult for attackers to access keys stored in the TEE with purely software means, therefore adding a first layer of protection for keys stored in the MCU.

In the context of secure binding, the TEE can be used to protect the 128-bit AES binding key-set necessary to establish the secure SCP03 channel between the MCU and EdgeLock SE05x and in this way prevent rogue applications in the REE to access the key-set. Moreover, EdgeLock SE05x should only be made accessible from within the TEE for additional security.

A TrustZone®-enabled MCU could be used to improve the security of the binding described in Section 2.2 as shown in Figure 8 . In this case the SCP03 channel is established by a trusted application from within the TEE. In this configuration, the EdgeLock SE05x Plug & Trust Middleware could be loaded in the TEE as shown in Figure 9 and used to manage the SCP channel establishment and the communication with EdgeLock SE05x. Future versions of EdgeLock SE05x Plug & Trust Middleware will also provide support for Arm Trusted Firmware-M (TF-M), a highly configurable set of software components to create a TEE that conforms to the Platform Security Architecture (PSA) specification. This will enable an even easier integration of EdgeLock SE05x software stack with TF-M compatible MCUs.

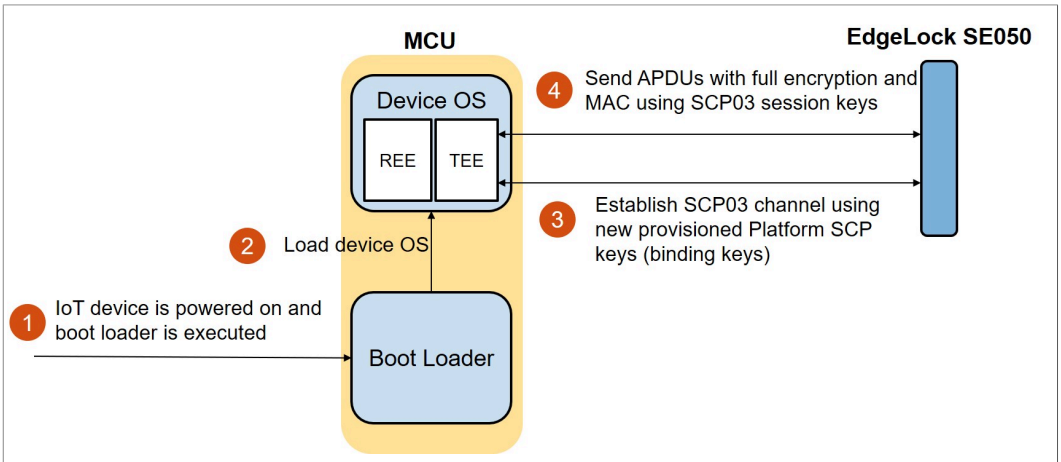

Figure 8. EdgeLock SE05x SCP channel establishment with TrustZone®

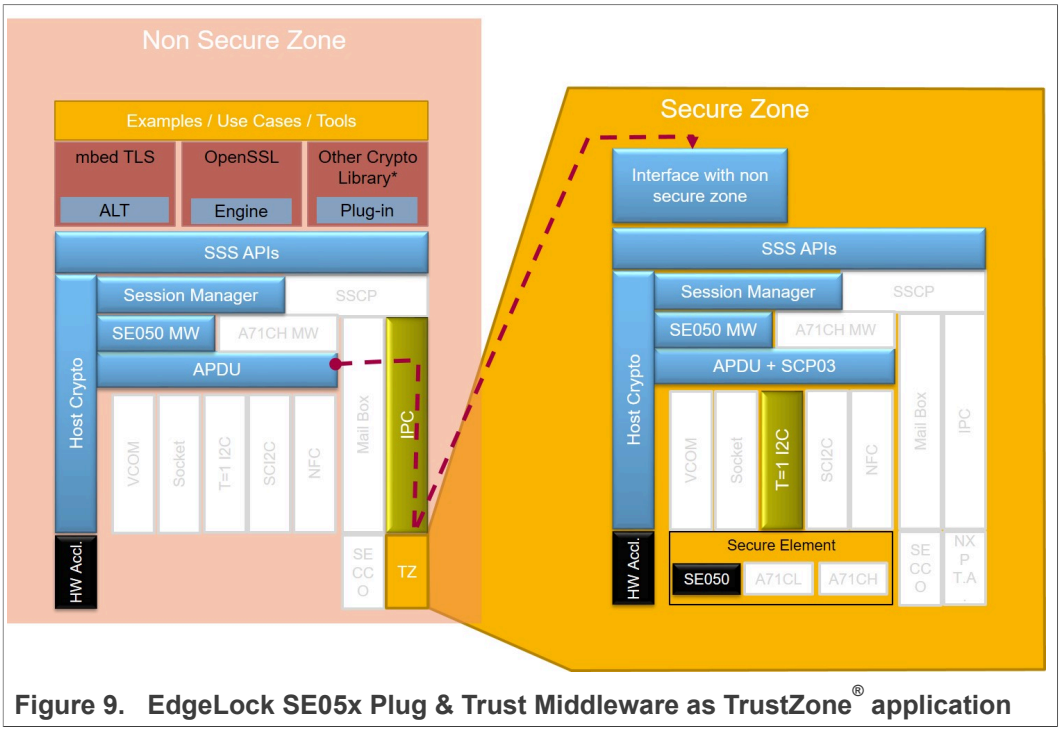

Figure 9. EdgeLock SE05x Plug & Trust Middleware as TrustZone® application

3.3 Binding for MCUs with TrustZone® and Cryptographic Acceleration and Assurance Module (CAAM)

CAAM is the cryptographic acceleration and assurance module included in NXP's i.MX MCUs. CAAM implements encryption, hashing and authentication algorithms and a secure memory controller among other related functionalities.

CAAM 's built-in blob protocol provides a method for protecting user-defined data across system power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Each time the blob protocol is used to protect data, a randomly generated key is used to encrypt the data. The random key is itself encrypted using an encryption key and is then stored along with the encrypted data. The encryption key is derived from the chip's master secret key so the encryption key can be recreated when the chip powers up again. The combination of encrypted random key and encrypted data is called a blob.

In the context of binding, the CAAM module can be used to generate the binding keys and to create a blob to securely store the keys. Since the blob is encrypted and bound to the particular MCU that generated it, it would be impossible for an attacker to extract the binding keys and use them in another device. SCP session keys would in this case be generated and retained inside the CAAM when the SCP channel is established and made accessible only to privileged software running in the TEE.

The initial binding process described in Section 2.2 would be modified as shown in Figure 10

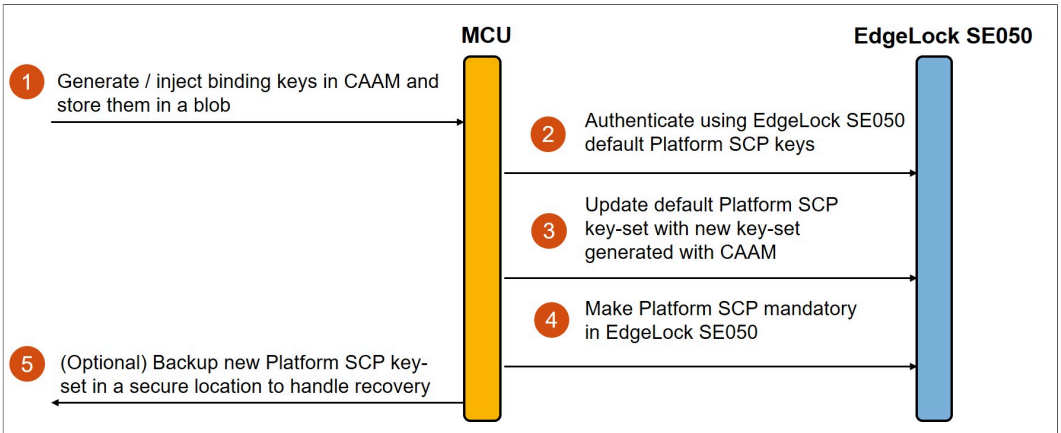

Figure 10. EdgeLock SE05x initial binding process using CAAM

Every time the device needs to use EdgeLock SE05x services, an SCP channel can be established using the new binding keys stored in the blob as shown in Figure 11:

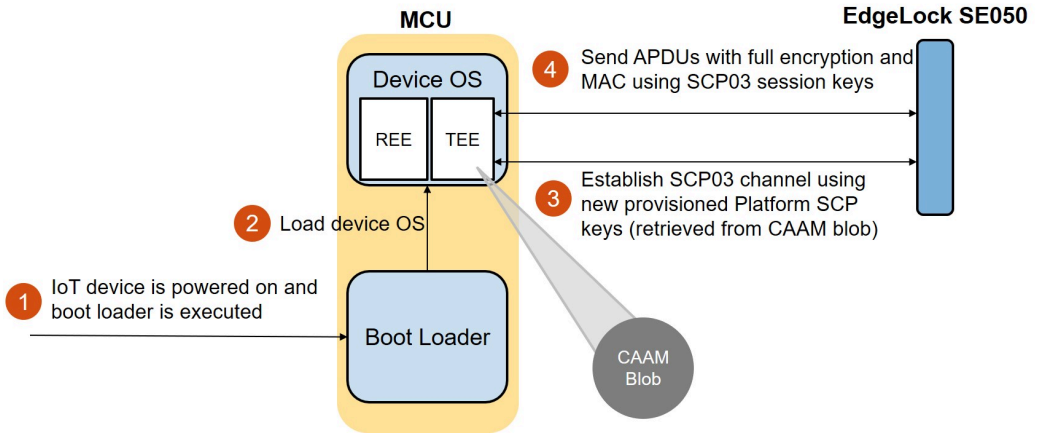

Figure 11. EdgeLock SE05x SCP channel establishment with TrustZone® and CAAM

3.4 Binding for MCUs with TrustZone® and Physically Unclonable Functions (PUFs)

TrustZone®-enabled MCUs already provide substantial security for storing binding keys. However, in use cases where top level security is of paramount importance, an additional layer of protection might be beneficial or even required.

In the TrustZone®-based setup described in Section 3.2, the binding keys are stored in a physical memory location whose access is protected by hardware and software means. Attackers with high skills and motivation could still be able to extract the binding keys by physically tampering with the IoT device, for example by deploying side channel attacks. To mitigate this risk, some advanced MCUs implement PUFs for the generation and retrieval of non-clonable, device-unique keys.

A Physically Unclonable Function (PUF) is a physical MCU component that for a given input and conditions returns a deterministic identifier of the device based on its physical characteristics. PUFs are most often based on unique physical variations which occur naturally during semiconductor manufacturing. The output of a PUF can be used as a unique identifier of the device. Such an identifier is by definition unclonable, since it is impossible to recreate the same microscopic physical variations that produce the PUF output.

MCUs implementing PUF typically provide hardware mechanisms to derive unique cryptographic keys from the unique identifier of the device. The general key derivation concept is depicted in Figure 12.

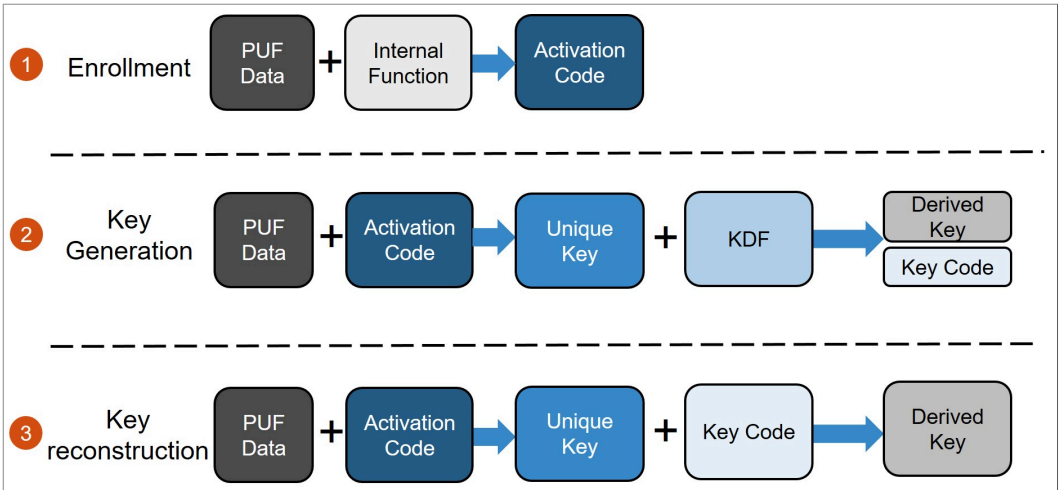

Figure 12. PUF enrollment, key generation and key reconstruction processes

1. **Enrollment:** Before starting to use a PUF, the PUF must be enrolled. This is a one-time process. The PUF data obtained from microscopic device variations is combined with hardware internal functions to generate a unique activation code which will be the device unique identifier. The activation code is public data and can be permanently stored in the PUF hardware registers;
2. **Key generation:** a unique fixed-length key is reconstructed by combining the PUF data with the activation code. The unique key is never stored in non-volatile memory. Typically the unique key is used as input of a Key Derivation Function (KDF). This allows the user to generate multiple diversified keys starting from one single key. The KDF might take as input some parameters, e.g. the derived key length. The KDF also generates a Key Code that can be later used to reconstruct the derived key. The Key Code can be stored permanently in the PUF hardware registers;
3. **Key reconstruction:** to reconstruct a previously generated derived key, the MCU unique key, obtained from the interpolation of the PUF data and the activation code, is combined with the Key Code. Derived keys are never stored in non-volatile memory locations.

In the context of binding, the process described above can be used to generate the SCP binding key-set and to reconstruct it every time the device is powered on. Keys can be removed from memory as soon as they are not needed anymore so to limit the time frame of attacks. Since the binding keys are not stored at any time in non-volatile memory, it becomes even more challenging for an attacker to extract the binding keys from the MCU. By taking advantage of TrustZone® to isolate hardware and software resources that have access to the PUF hardware, this configuration allows you to ensure a very high level of protection for binding keys.

The initial, one-time binding process described in Figure 3 in Section 2.2 can be adapted to support the PUF feature as shown in Figure 13. The establishment of the SCP channel remains mostly unchanged, but now binding keys can be retrieved from within the TrustZone® using the pre-injected activation code and Key Codes. The process is depicted in Figure 14.

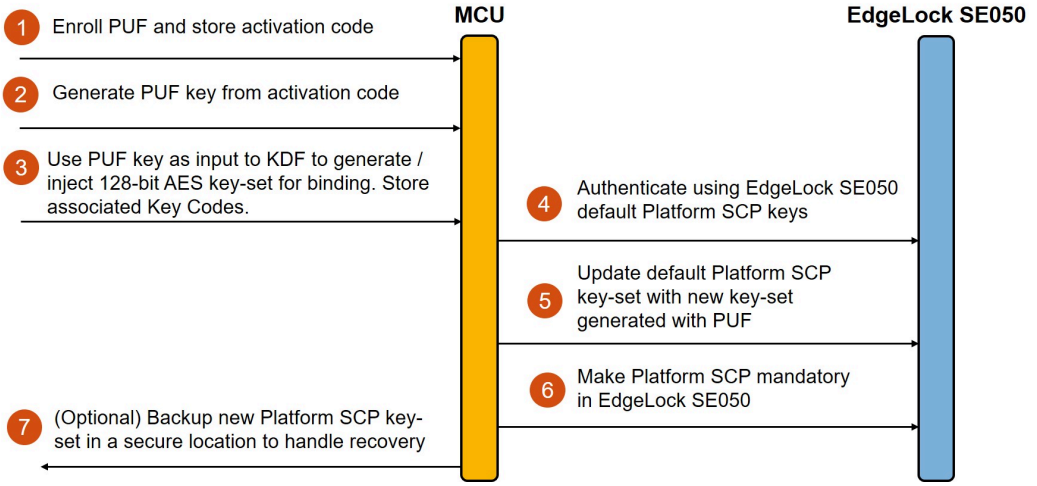

Figure 13. Initial binding process using PUF

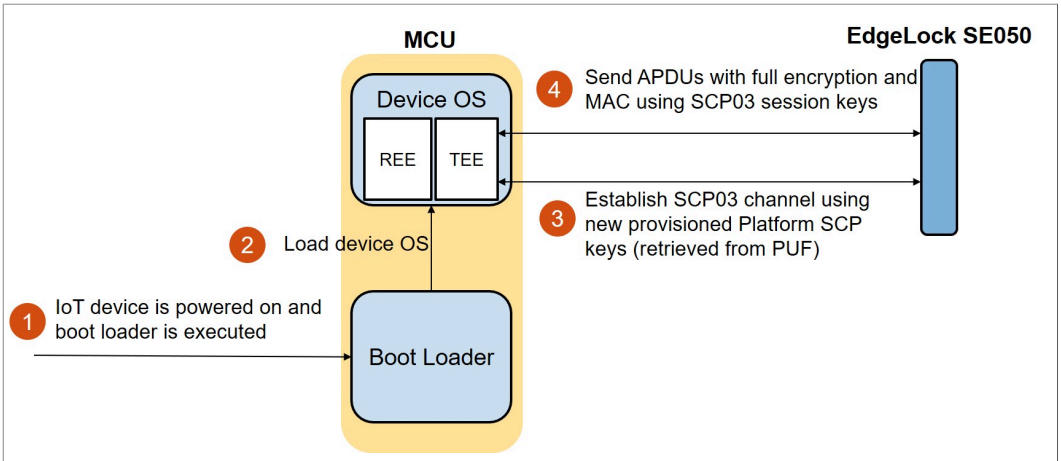

Figure 14. SCP channel establishment at device boot up using PUF keys

3.5 Binding for MCUs with TrustZone®, PUF and trust-provisioned keys

MCUs with TrustZone® and PUF key generation capabilities guarantee a strong and reliable protection for SCP binding keys. However, to obtain the maximum possible level of security, the generation and injection of the binding keys in the MCU and in the SE must be performed in a secure environment using well-established secure processes. This limits the attack perimeter and significantly reduces the risks of a potential leakage of the binding keys in comparison with a post-deployment injection scenario.

The injection of the binding keys should be performed as early as possible in the manufacturing process to avoid the technical and financial costs of maintaining secure processes at multiple manufacturing stages. For this reason, it is typically more convenient to bundle the MCU and the SE in a single module and perform the injection of the binding keys in the trusted, secure testing facilities of the module manufacturer. The injection process described in Section 2.2 would therefore change as shown in Figure 15.

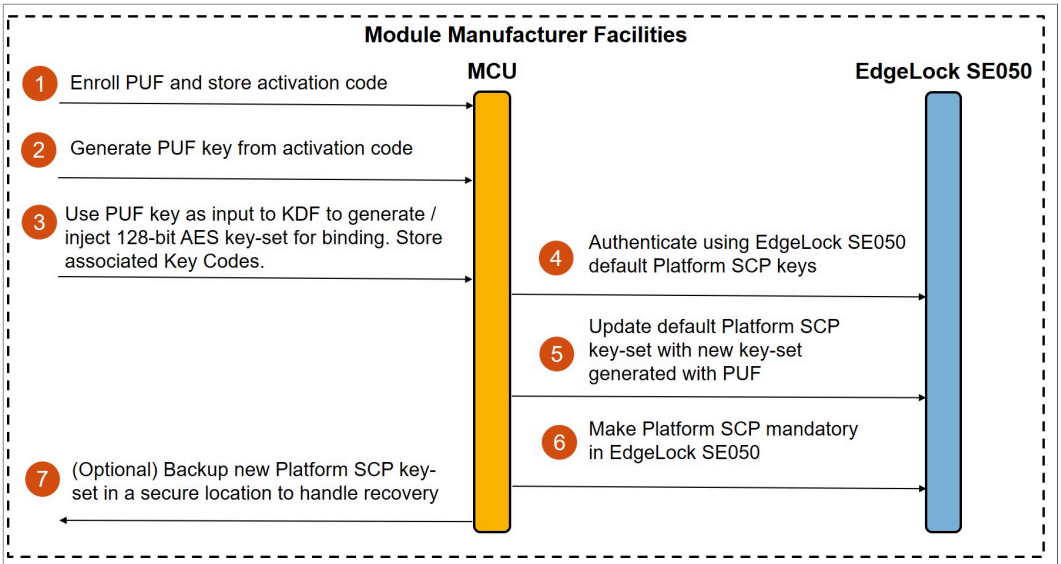

Figure 15. Initial binding process with trust-provisioned keys

In this case, the module manufacturer would be responsible to perform the whole initial binding process, including the generation of the binding keys and their injection in EdgeLock SE05x. The optional backup of the binding keys is performed by the module manufacturer and can then be shared with the device manufacturer using secure processes. Once the module is shipped, the device manufacturer would only need to establish the SCP session using the pre-injected binding keys.

Note: the availability and conditions of this option must be verified beforehand with the module manufacturer.

4 Appendix: Secure Boot

Binding can be performed during the boot process of the IoT device to harden the connection between EdgeLock SE05x and the MCU.

Secure Boot is a feature implemented in some MCUs that ensures that only a verified, signed boot loader, firmware or OS is executed when the IoT device is powered on. With secure boot in place, it would be even more difficult for an attacker to tamper with the boot process. In the context of binding, Secure Boot helps ensuring that the SCP binding is performed by a verified, trusted software component such as the device boot loader.

Secure Boot relies on a pre-injected public key that is typically stored in a MCU read-only area (e.g. ROM). This public key effectively establishes a chain of trust at the lowest possible hardware level and it is used to ensure the integrity and authenticity of the boot process. Starting with an implicitly trusted component, e.g. the MCU, every stage of the boot chain can be authenticated by verifying the signature of the code before it is loaded and executed. The ownership of the trust chain can change at each stage, so for example, a Secondary Boot Loader (SBL) can be validated using a public key injected by the chip manufacturer in the ROM boot loader, while the final OS image can be validated using a public key provisioned by the device manufacturer in the SBL.

An example of Secure Boot architecture that includes EdgeLock SE05x is shown in Figure 16. In this example architecture, SCP binding is performed in the SBL when the device boots up.

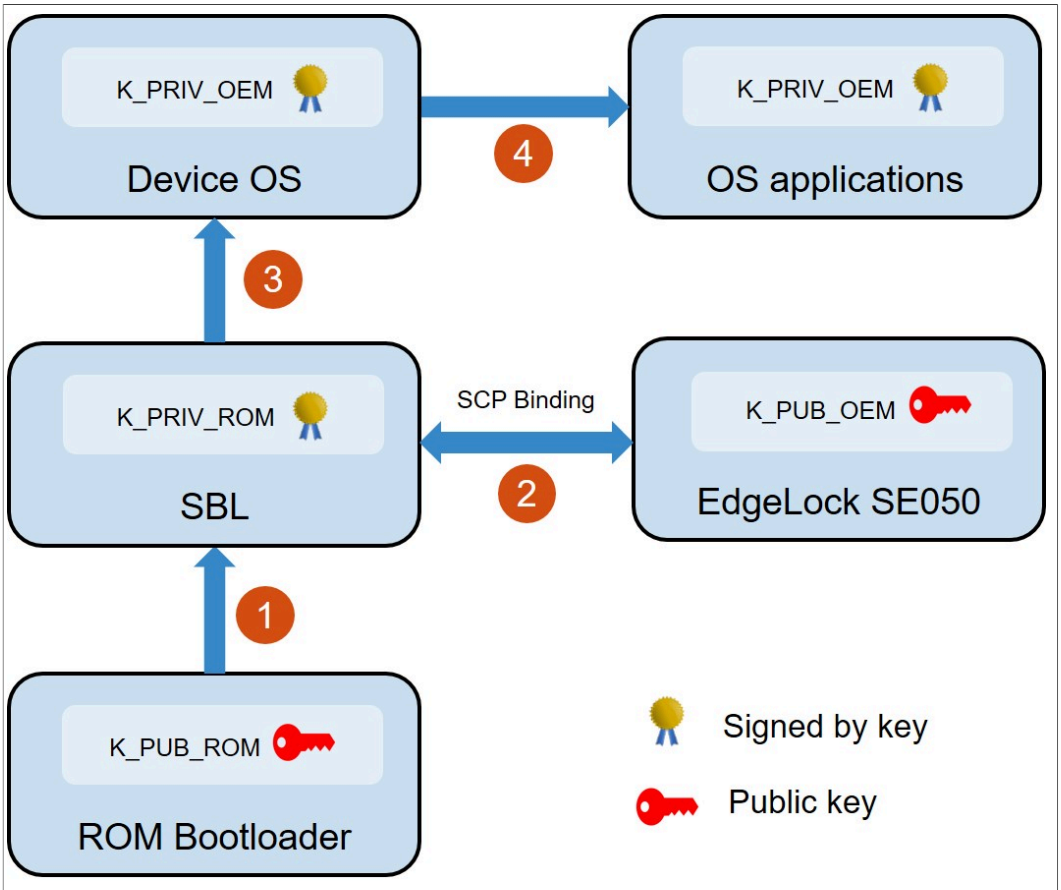

Figure 16. Secure Boot example using EdgeLock SE05x

1. The ROM boot loader uses the public key that was pre-injected by the chip manufacturer (K_PUB_ROM) to verify the authenticity of the SBL before loading it. The SBL must therefore be signed with the corresponding private key (K_PRIV_ROM) owned by the chip manufacturer. Since in this scenario the SBL is owned by the device manufacturer, signing the SBL might require an interaction with the chip manufacturer;
2. The SBL is loaded and the SCP binding with EdgeLock SE05x is performed. The ownership of the secure boot trust chain can therefore securely be transferred to EdgeLock SE05x. The public key of the device manufacturer pre-injected in EdgeLock SE05x (K_PUB_OEM) is used to authenticate the remaining steps of the boot process;
3. The SBL finally loads the Device OS image after validating its signature using the K_PUB_OEM stored in EdgeLock SE05x. The device OS image must be signed with the corresponding private key owned by the device manufacturer (K_PRIV_OEM);
4. For additional security, OS applications can also be validated with K_PUB_OEM before execution. In this case only applications signed by the device manufacturer could be executed in the OS.

5 Legal information

5.1 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.

5.2 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.

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.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a 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 or documented vulnerabilities. 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.

5.3 Trademarks

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

Figures

Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 7 December 2020
Document identifier: AN12662