AN1271: Secure Key Storage

Secure Key Storage is a feature in Secure Vault High devices that allows for the protection of cryptographic keys by key wrapping. User keys are encrypted by the device's root key for non-volatile storage for later usage. This prevents the need for a key to be stored in plaintext format on the device, preventing attackers from gaining access to the keys through traditional flash-extraction or application attacks, and allowing for a potentially unlimited number of keys to be securely stored in any available storage.

Series 2 devices can use TrustZone to implement Secure Key Storage, so this feature is now also available on Secure Vault Mid devices.

This document describes the operation and usage of this feature, and provides comparisons with other key storage methods.

KEY POINTS

- Keys are encrypted or 'wrapped' with a Secure Engine root key
- Secure Engine root key is not stored on the device, instead it is generated on each reset
- Wrapped keys are confidential to the Secure Engine, and can be stored in non-volatile memory safely
- Wrapped keys can be cached into Secure Engine for usage at a later time
- TrustZone Secure Key Storage

1. Series 2 Device Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Series 2 products that included a Secure Engine. The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys and to execute cryptographic functions and secure services.

On Series 1 devices, the security features are implemented by the TRNG (if available) and CRYPTO peripherals.

On Series 2 devices, the security features are implemented by the Secure Engine and CRYPTOACC (if available). The Secure Engine may be hardware-based, or virtual (software-based). Throughout this document, the following abbreviations are used:

- HSE - Hardware Secure Engine
- VSE - Virtual Secure Engine
- SE - Secure Engine (either HSE or VSE)

Additional security features are provided by Secure Vault. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

Level (1)	SE Support	Part (2)
Secure Vault High (SVH)	HSE only (HSE-SVH)	Refer to UG103.05 for details on supporting devices.
Secure Vault Mid (SVM)	HSE (HSE-SVM)	"
"	VSE (VSE-SVM)	"
Secure Vault Base (SVB)	N/A	"

Note:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. UG103.05.

Secure Vault Mid consists of two core security functions:

- Secure Boot: Process where the initial boot phase is executed from an immutable memory (such as ROM) and where code is authenticated before being authorized for execution.
- Secure Debug access control: The ability to lock access to the debug ports for operational security, and to securely unlock them when access is required by an authorized entity.

Secure Vault High offers additional security options:

- Secure Key Storage: Protects cryptographic keys by “wrapping” or encrypting the keys using a root key known only to the HSE-SVH.
- Anti-Tamper protection: A configurable module to protect the device against tamper attacks.
- Device authentication: Functionality that uses a secure device identity certificate along with digital signatures to verify the source or target of device communications.

A Secure Engine Manager and other tools allow users to configure and control their devices both in-house during testing and manufacturing, and after the device is in the field.

1.1 User Assistance

In support of these products Silicon Labs offers whitepapers, webinars, and documentation. The following table summarizes the key security documents:

Document	Summary	Applicability
AN1190: Series 2 Secure Debug	How to lock and unlock Series 2 debug access, including background information about the SE	Secure Vault Mid and High
AN1218: Series 2 Secure Boot with RTSL	Describes the secure boot process on Series 2 devices using SE	Secure Vault Mid and High
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using SE during device production	Secure Vault Mid and High
AN1247: Anti-Tamper Protection Configuration and Use	How to program, provision, and configure the anti-tamper module	Secure Vault High
AN1268: Authenticating Silicon Labs Devices using Device Certificates	How to authenticate a device using secure device certificates and signatures, at any time during the life of the product	Secure Vault High
AN1271: Secure Key Storage (this document)	How to securely “wrap” keys so they can be stored in non-volatile storage.	Secure Vault High

1.2 Key Reference

Public/Private keypairs along with other keys are used throughout Silicon Labs security implementations. Because terminology can sometimes be confusing, the following table lists the key names, their applicability, and the documentation where they are used.

Key Name	Customer Programmed	Purpose	Used in
Public Sign key (Sign Key Public)	Yes	Secure Boot binary authentication and/or OTA upgrade payload authentication	AN1218 (primary), AN1222
Public Command key (Command Key Public)	Yes	Secure Debug Unlock or Disable Tamper command authentication	AN1190 (primary), AN1222, AN1247
OTA Decryption key (GBL Decryption key) aka AES-128 Key	Yes	Decrypting GBL payloads used for firmware upgrades	AN1222 (primary), UG266/UG489
Attestation key aka Private Device Key	No	Device authentication for secure identity	AN1268

1.3 SE Firmware

Silicon Labs strongly recommends installing the latest SE firmware on Series 2 devices to support the required security features. Refer to AN1222 for the procedure to upgrade the SE firmware and UG103.05 for the latest SE Firmware shipped with Series 2 devices and modules.

2. Introduction

The HSE isolates cryptographic functions and data from the host Cortex-M33 core. It is used to accelerate cryptographic operations as well as to provide a method to securely store keys. This application note will cover the Secure Key Storage feature of the HSE-SVH devices.

The HSE contains one-time programmable memory (OTP) key storage slots for three specific keys:

1. The Public Sign Key, used for Secure Boot and Secure Upgrades
2. The Public Command Key, used for Secure Debug unlock and tamper disable
3. The Symmetric OTA Decryption Key, used for Over-The-Air updates

These keys are one-time programmable, and, after programming, are persistent for the lifetime of the device.

HSE-SVH devices also contain four volatile storage slots for any other user keys. These slots are not persistent through a reset. In the case where a key needs persistent storage, the key must be stored outside of the HSE in non-volatile storage. After a device reset, the key can be loaded into the HSE volatile key storage for usage by index, or used in-place (passed to the HSE on every requested operation). Without any secure key storage mechanism, the user key stored in non-volatile storage is opened to storage-extraction attacks (such as gaining access to and downloading device flash), as well as application-level attacks (i.e., taking control of the user application or privileges in a manner that allows access to the keys).

With Secure Key Storage, a user can only access a key from the HSE in a 'wrapped' format. In this format, the key is encrypted by a device-unique root key, only available to the HSE. This allows a user to store a key confidentially in non-volatile storage to provide key persistence. Using Secure Key Storage, the plaintext key is never stored in non-volatile memory, preventing storage-extraction attacks from obtaining the key. After a device reset, the wrapped key can be loaded into the HSE for usage without ever exposing the plaintext key to the application, which also prevents application-level attacks from exposing the key.

SVM devices can only support Secure Key Storage through the use of TrustZone. GSDK v4.2.2 is the first version to support TrustZone software development on Series 2 devices.

Silicon Labs provides Custom Part Manufacturing Service (CPMS) to inject custom secret keys on the chips during manufacturing. For more information about CPMS, see UG519: Custom Part Manufacturing Service User's Guide.

3. HSE Secure Key Storage

The following sections demonstrate three methods for key storage: ARM® TrustZone®, plaintext, and Secure Key Storage.

Note: In the following examples, AES key usage is demonstrated. However, any other key types supported by the device can also be used for key storage.

3.1 Key Generation and Usage

In HSE-SVH devices, cryptographic functions are performed by the HSE. In order to perform these functions, the HSE must have access to any user keys needed. Keys can be generated and used by the HSE in multiple ways:

1. External storage, in-place usage:
   a. A user generates a plaintext key and stores it in device memory.
   b. The user provides a key descriptor to the HSE that points to this key for a specific cryptographic operation.
   c. The HSE performs the cryptographic operation using this key, but does not store it in any HSE volatile storage slot.
2. External storage with HSE import:
   a. A user generates a plaintext key and stores it in device memory.
   b. The user provides a key descriptor to the HSE that points to this key, as well as a slot number to store the key.
   c. The HSE imports this key into a volatile key storage slot or can optionally save it in wrapped form in device memory.
   d. The user requests that the HSE performs a cryptographic function by providing the index of the storage slot or a pointer to the wrapped key in device memory.
3. Internal HSE key generation:
   a. The user commands the HSE to generate a new key within one of the HSE's volatile key slots or can optionally save it in wrapped form in device memory.
   b. The user requests that the HSE performs a cryptographic function by providing the index of the storage slot or a pointer to the wrapped key in device memory.

Note:

- In each case, to provide persistent storage for the key, the key must be stored in non-volatile memory.
- 3.2 Plaintext Key Storage and 3.3 Secure Key Storage provide details on key generation and usage with HSE-SVH device.

3.2 Plaintext Key Storage

3.2.1 Plaintext Key Import

The simplest manner to store a key is to save it in plaintext form. The steps to store and use a key stored in plaintext form are as follows:

1. A user key is generated and imported into device memory. For persistent storage, this must be non-volatile storage, such as device flash.
2. After a device reset, the HSE volatile key storage will be empty. The plaintext key is imported (method 2) into a slot for usage. Alternatively, the key could be used in place (method 1) from non-volatile storage on a per-operation basis.

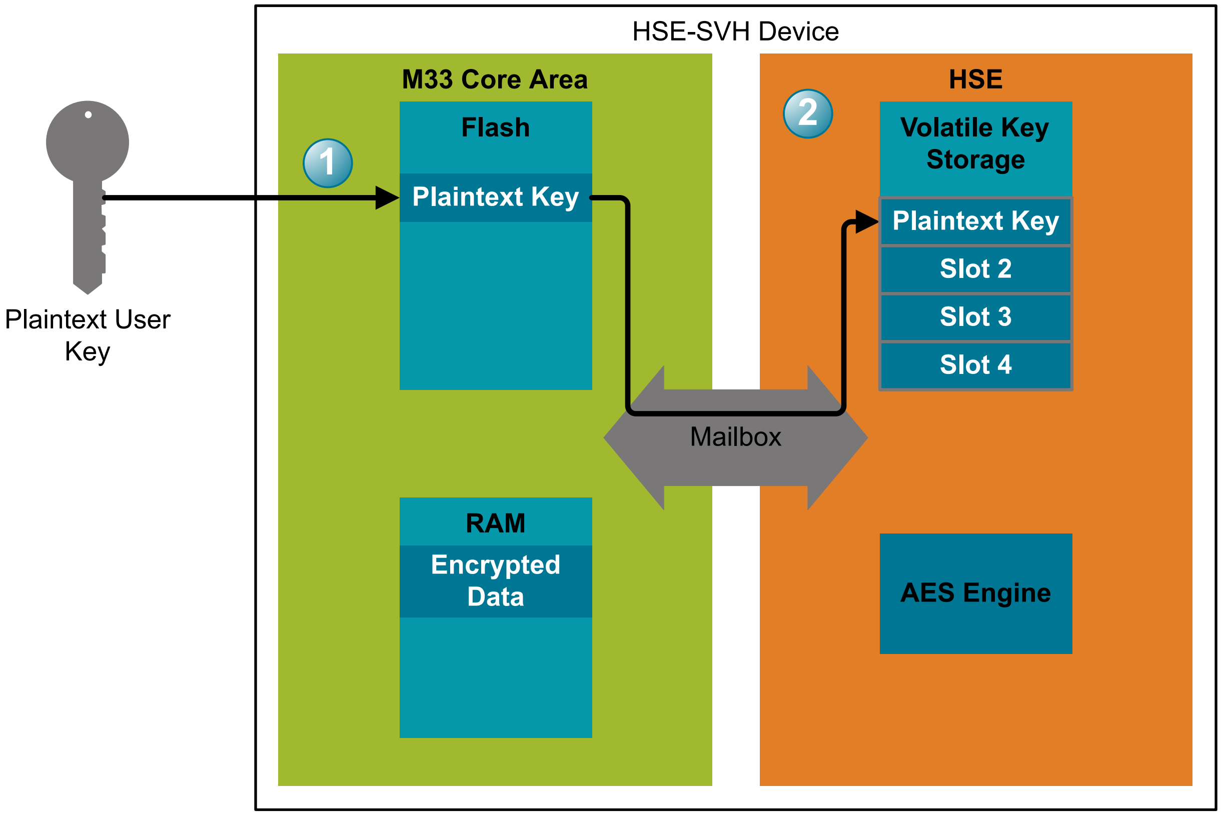


Figure 3.1. Plaintext Key Import

3.2.2 Plaintext Key Usage

In order to use the key for a cryptographic operation, the following procedure is used.

1. The user passes data to be processed (in this specific example, AES encrypted data) to the HSE.
2. The user requests that a cryptographic operation be performed on this data using one of the keys stored in the HSE volatile key storage slots (method 2). Alternatively, the key can be passed to the HSE directly for a singular cryptographic operation (method 1).
3. The HSE performs the cryptographic operation.
4. The output of the cryptographic operation is passed back to the user for processing.

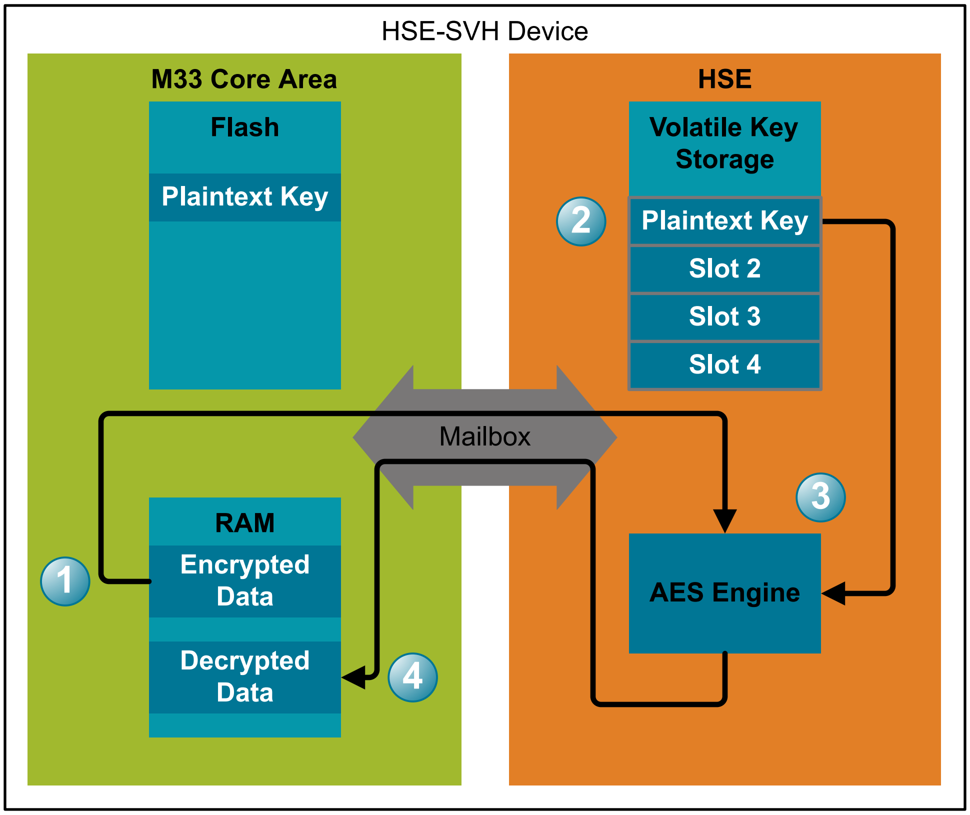


Figure 3.2. Plaintext Key Usage

This method exposes the keys to two major vulnerabilities:

1. Access to device storage gives access to the keys. In this case, an attack that gains access to the flash contents will expose the user key.
2. Since the application has access to the keys, compromising the application or device privileges can compromise the keys. Such an attack might not directly access device memory, but take control of the application in a way that causes the application to expose the key to an attacker.

3.3 Secure Key Storage

With Secure Key Storage, the user key, using the HSE, can be accessed in an encrypted, or 'wrapped' form. Only the HSE has access to the HSE root key used to decrypt, or 'unwrap', the wrapped key. This HSE root key is not stored on the device during power-down, but rather reconstructed after each reset. Key wrapping allows a user to securely store a key in non-volatile memory, limiting the number of keys that can be stored only by the amount of storage the user has available.

Note: The reconstructed root key after each reset is IDENTICAL and UNIQUE on each HSE-SVH device.

3.3.1 Wrap an External Key

To wrap an externally-generated key:

1. After power-on, the device's unique root key is reconstructed with output from the Physically Unclonable Function (PUF).
2. A user key is generated and imported into device memory. In this example, the key is imported into RAM for easy deletion, and the added security that, if device power is removed, the key will be lost.
3. The user key is passed to the HSE, where it is encrypted with the HSE's root key.
4. The wrapped key is passed back to the user application for storage in non-volatile memory (in this case, device flash).
5. The plaintext key can now be deleted from the device. From this point forward, only the HSE will have access to the plaintext key.

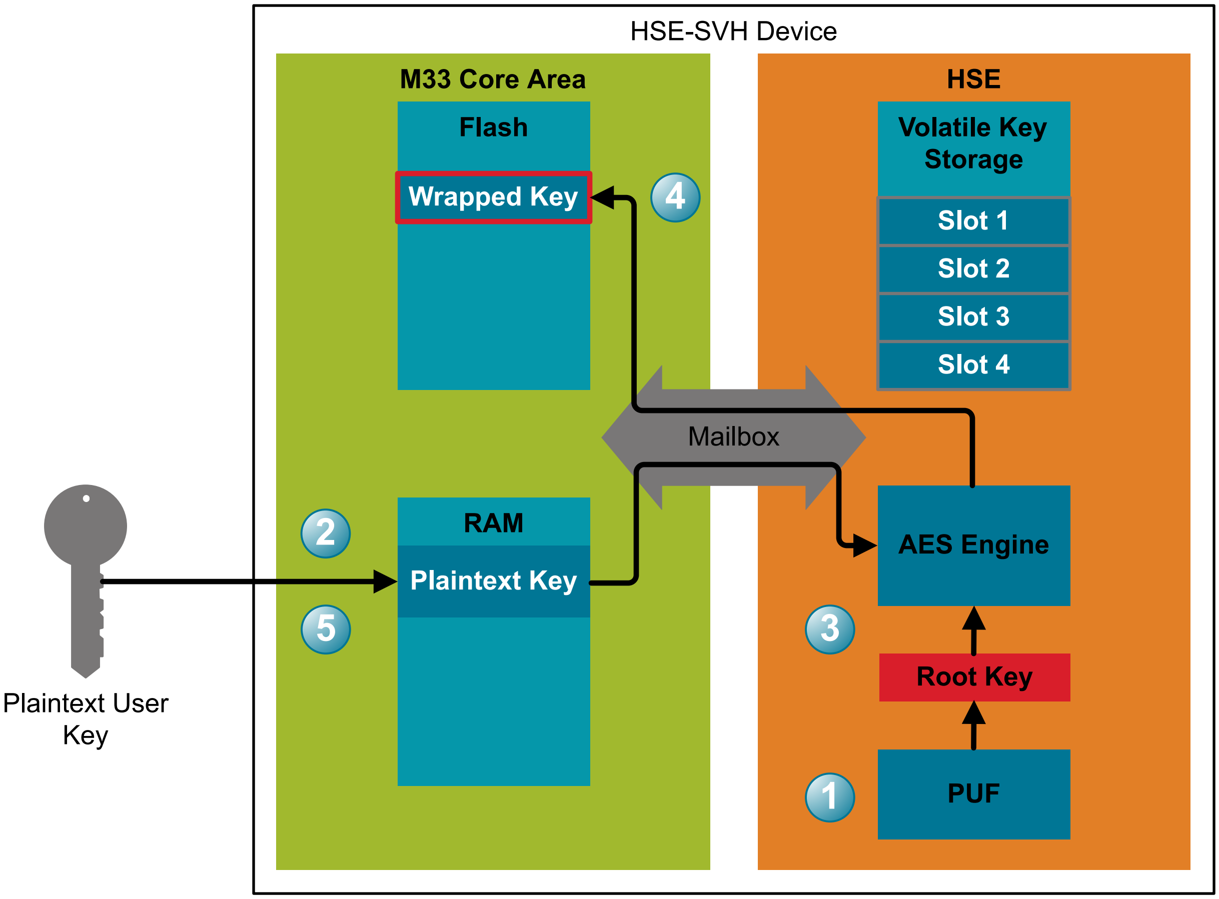


Figure 3.3. External Key Import, Wrapping, and Storage

3.3.2 Generate an Internal Wrapped Key

Instead of importing an external key, the HSE can generate a new key directly into one of its volatile key storage slots. This key can then be exported in wrapped form for secure persistent storage.

1. The user requests that the HSE generates a new key into one of its storage slots using the True Random Number Generator (TRNG).
2. The key is encrypted with the HSE's root key.
3. The wrapped key is passed back to the user application for non-volatile storage (flash, in this case).

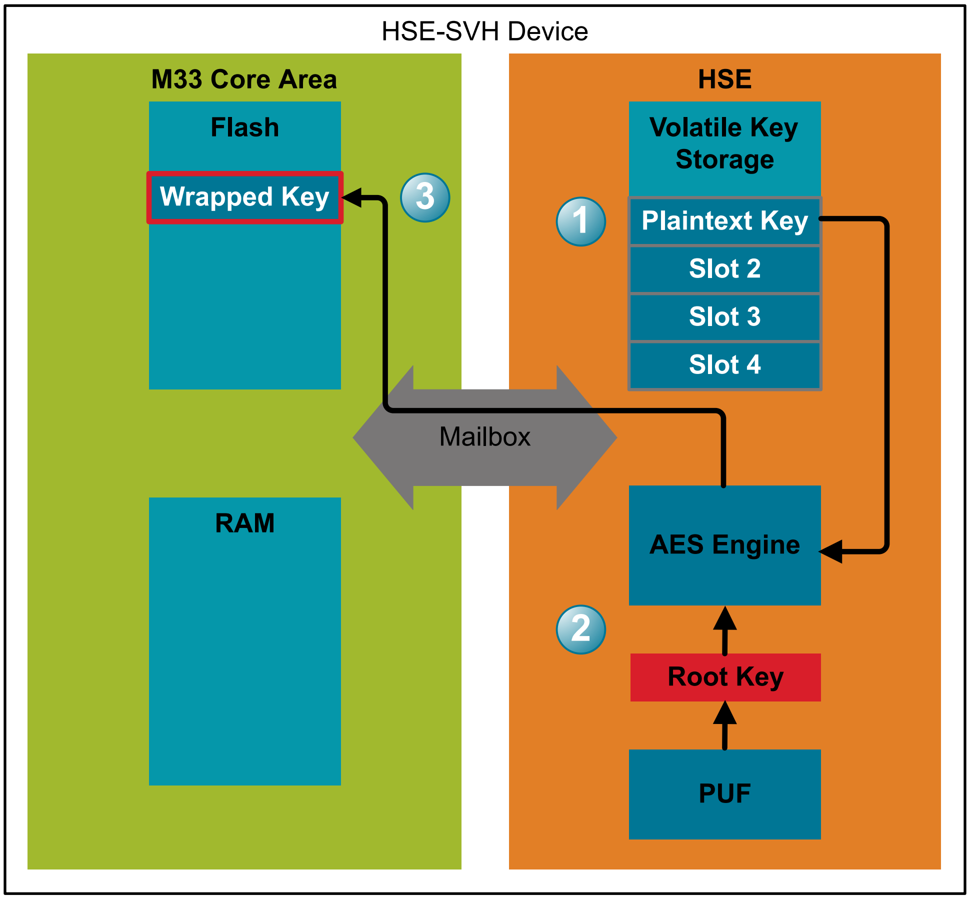


Figure 3.4. Internally Generated Key Wrapping and Storage

3.3.3 Wrapped Key Import

In order to import a wrapped key into the HSE for usage:

1. The wrapped key is passed to the HSE.
2. The wrapped key is decrypted ("unwrapped") with the HSE's root key.
3. The plaintext key is stored in a volatile key storage slot.

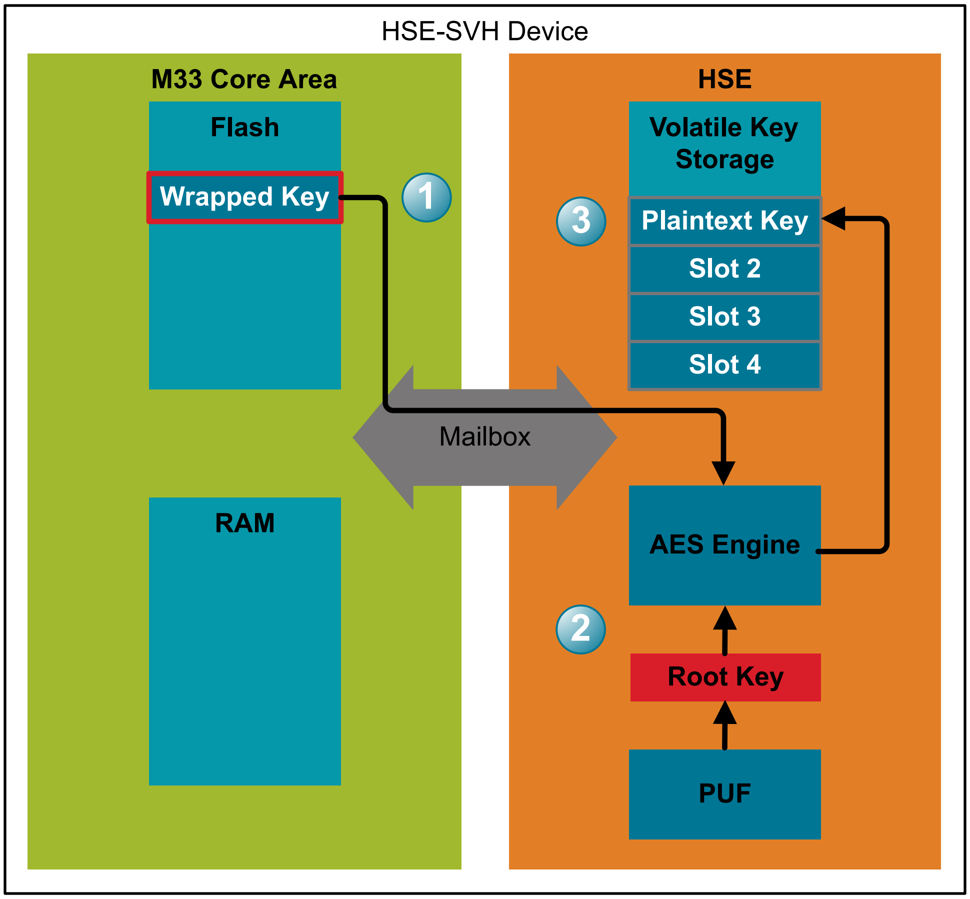


Figure 3.5. Wrapped Key Import

3.3.4 Wrapped Key Usage

In order to use the key for a cryptographic operation, the same steps are followed as when using a plaintext key that has been imported into the HSE:

1. The user passes data to be processed (in this specific example, AES encrypted data) to the HSE.
2. The user requests that a cryptographic operation be performed on this data using one of the keys stored in the HSE volatile key storage slots. Alternatively, the wrapped key can be passed to the HSE directly for a singular cryptographic operation. In this case, the key will be unwrapped before being used, but will not be stored for future operations.
3. The HSE performs the cryptographic operation.
4. The output of the cryptographic operation is passed back to the user for processing.

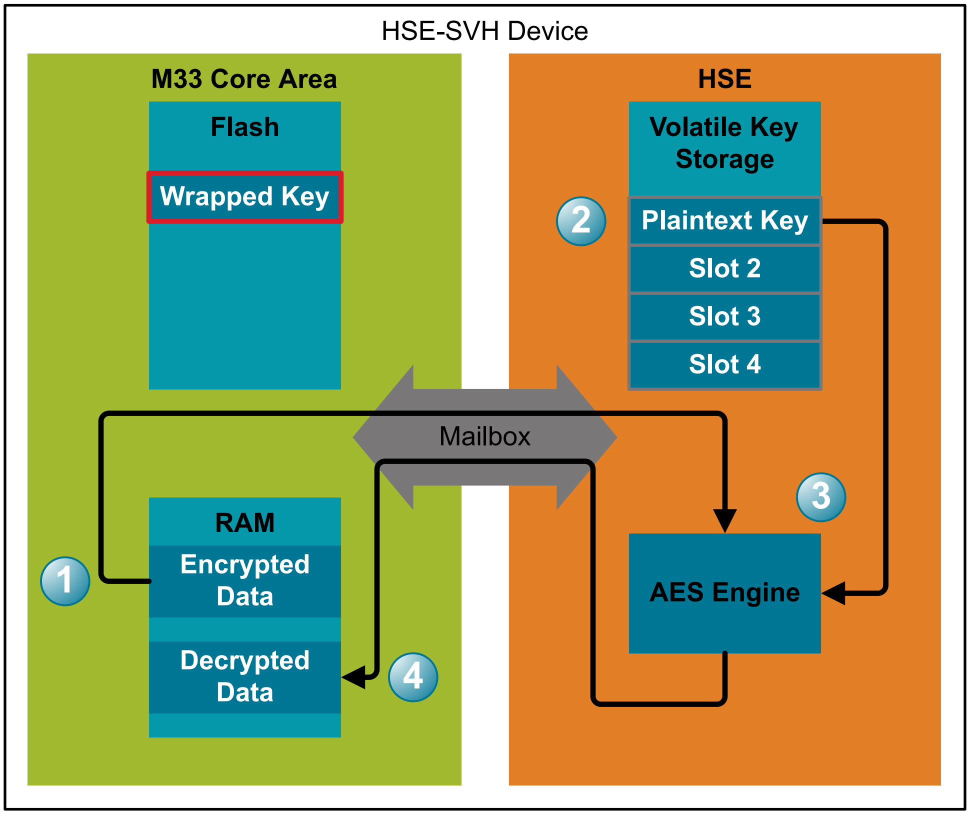


Figure 3.6. Wrapped Key Usage

3.4 Secure Key Storage Advantages

Secure Key Storage confers the following benefits over other key storage methods:

1. Access to device memory does not expose user keys.
2. Compromising the user application does not expose user keys, since the user application itself does not have access to the plaintext keys.
3. The number of user keys that can be securely stored is only limited by the amount of storage available to the user, including external storage.

3.5 Operation Details

3.5.1 Root Key Generation

Secure Key Storage depends on the HSE to encrypt / decrypt (wrap / unwrap) user keys with its own symmetric root key. The symmetric key used for this wrapping and unwrapping must be highly secure as it can expose all other key material in the system. The HSE key Management system uses a Physically Unclonable Function (PUF) to generate a persistent device-unique seed on power up to dynamically reconstruct this critical root key. The key is only visible to the AES encryption engine, and it is not retained when the device loses power.

3.5.2 Access a Wrapped Key

By default, a key in an HSE storage slot can be exported to the application as a plaintext key. To prevent this, the user can use the key descriptor to set a user key to non-exportable. This option prevents any request to export the wrapped key in plaintext from HSE, so the user application can only access the key encrypted by the HSE's root key. The HSE also tags the key with information to identify the wrapped key. Since only the HSE can access the root key to unwrap the user key, the plaintext key is non-accessible to the user application.

Note: Wrapped keys are slightly larger than the equivalent plaintext key, as some additional metadata is required to identify the wrapped key to the HSE.

3.5.3 Wrapped Key Storage and Usage

Once a key has been wrapped, it can be safely stored anywhere - device flash, RAM, external storage, etc. The number of keys that can be securely stored is only limited by the available storage space. A wrapped key can later be imported into a HSE volatile storage slot for usage, or used in-place. Once the key is wrapped and stored, the plaintext key available to the application can be deleted. From here, only the HSE will have the ability to unwrap and use the key.

With access to the wrapped key, the HSE can use this key in one of two ways:

1. A user can request that a cryptographic operation be performed using the key stored in memory. In this case, the HSE will import the key, unwrap it, and then perform the cryptographic operation. The key will not be stored within the HSE.
2. A user can import the wrapped key into a HSE volatile storage slot. In this case, the key is unwrapped by the HSE and stored in plaintext in a volatile slot. The user can then later request that a cryptographic function be performed by the HSE by referencing the volatile slot index. This provides a performance increase over using wrapped keys in place, as the HSE does not need to import and unwrap the key on each requested operation.

3.5.4 Password Protection

When defining a key descriptor for a new key, or when importing an existing key into HSE, the user can choose to require a password to allow use of the key. The password field in the key descriptor structure is eight bytes in length. If unspecified, the key will use the default password of all zeros.

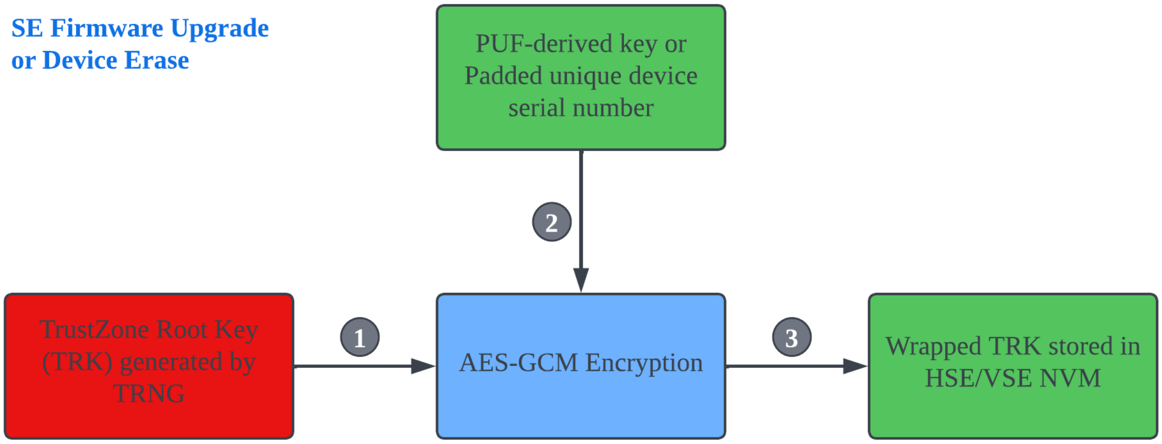
After importing a key with a password, failing to provide the correct password when performing a cryptographic operation will result in HSE returning an invalid credentials error, and no operation will be performed.

4. TrustZone Secure Key Storage

In Series 2 devices, key management can be handled by a feature called TrustZone. TrustZone divides the device memory map into a Secure Processing Environment (SPE) and a Non-secure Processing Environment (NSPE). User code is executed from the NSPE, which cannot access any part of the SPE. The SPE is used to store cryptographic keys securely and to control other Secure operations.

The following sections describe using TrustZone on Series 2 devices for Secure Key Storage. Refer to AN1374 for details about TrustZone implementation on Series 2 devices.

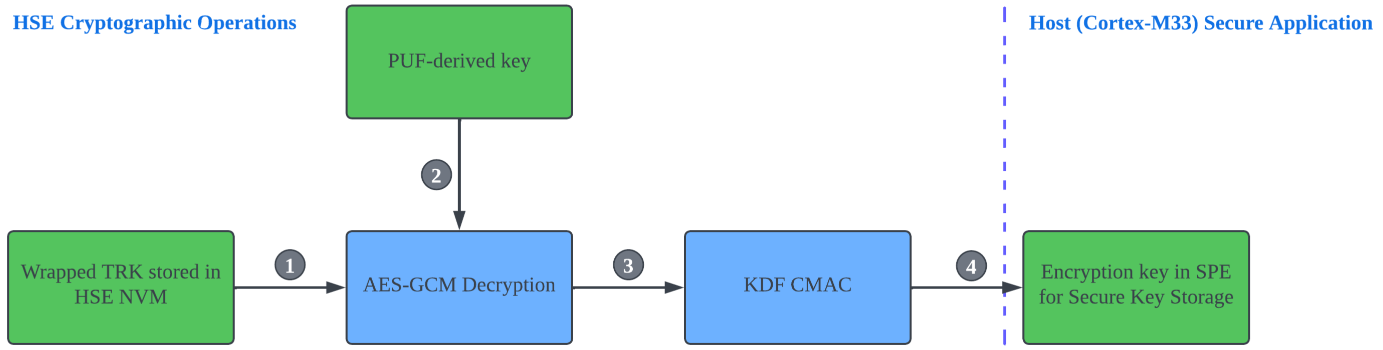
4.1 TrustZone Root Key Generation (HSE and VSE)


SE Firmware Upgrade
or Device Erase
PUF-derived key or Padded unique device serial number
2
TrustZone Root Key (TRK) generated by TRNG
1
AES-GCM Encryption
3
Wrapped TRK stored in HSE/VSE NVM

1. The TrustZone Root Key (TRK) is generated by the True Random Number Generator (TRNG) in Series 2 devices.
2. The PUF-derived key (HSE and xG27 VSE devices) or padded unique device serial number (xG22 VSE devices) is used to wrap (AES-GCM) the TRK.
3. The wrapped TRK is stored in the SE Non-volatile memory (NVM), and the TRK in RAM is deleted.
    - The wrapped TRK already existed if the shipped Series 2 device with SE firmware version supports this key.
    - The wrapped TRK will be generated when upgrading from a SE firmware version that did not support this key to the one that does.
    - The wrapped TRK will be renewed after performing a Device Erase.

Note: The Physically Unclonable Function (PUF) is not retained when the device loses power, so the TRK wrapped by the PUF-derived key is not vulnerable to a storage-extraction attack.

4.2 TrustZone Root Key Usage (HSE)

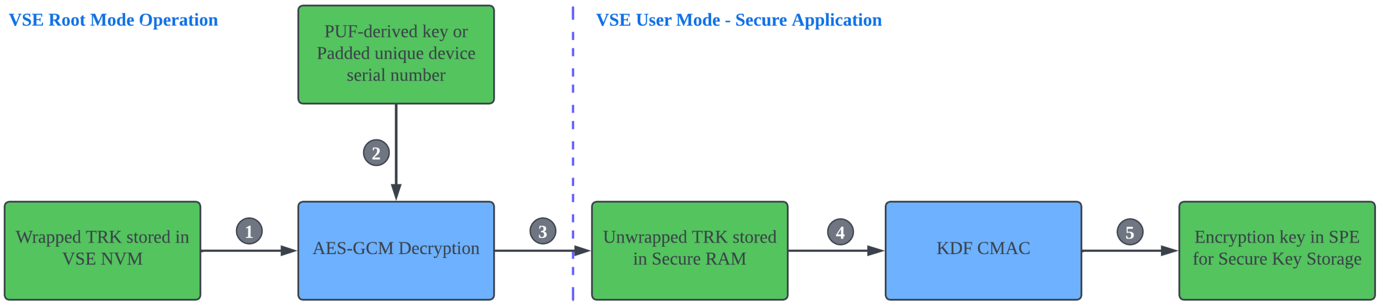

HSE Cryptographic Operations
Host (Cortex-M33) Secure Application
PUF-derived key
2
Wrapped TRK stored in HSE NVM
1
AES-GCM Decryption
3
KDF CMAC
4
Encryption key in SPE for Secure Key Storage

1. The Secure application in the host uses a non-exportable built-in key to access the wrapped TRK in HSE NVM for cryptographic operations.
2. The PUF-derived key is used to decrypt (AES-GCM) the wrapped TRK in HSE NVM.
3. The unwrapped TRK in the HSE is the master key of a Key Derivation Function (KDF).
4. The encryption key in SPE for Secure Key Storage is derived from the KDF CMAC.

Note:

- All cryptographic operations are performed by the HSE (security co-processor).
- Only the HSE can access the unwrapped TRK for KDF, so this key will not expose the Secure application in the host.

4.3 TrustZone Root Key Usage (VSE)


VSE Root Mode Operation
VSE User Mode - Secure Application
PUF-derived key or Padded unique device serial number
2
Wrapped TRK stored in VSE NVM
1
AES-GCM Decryption
3
Unwrapped TRK stored in Secure RAM
4
KDF CMAC
5
Encryption key in SPE for Secure Key Storage

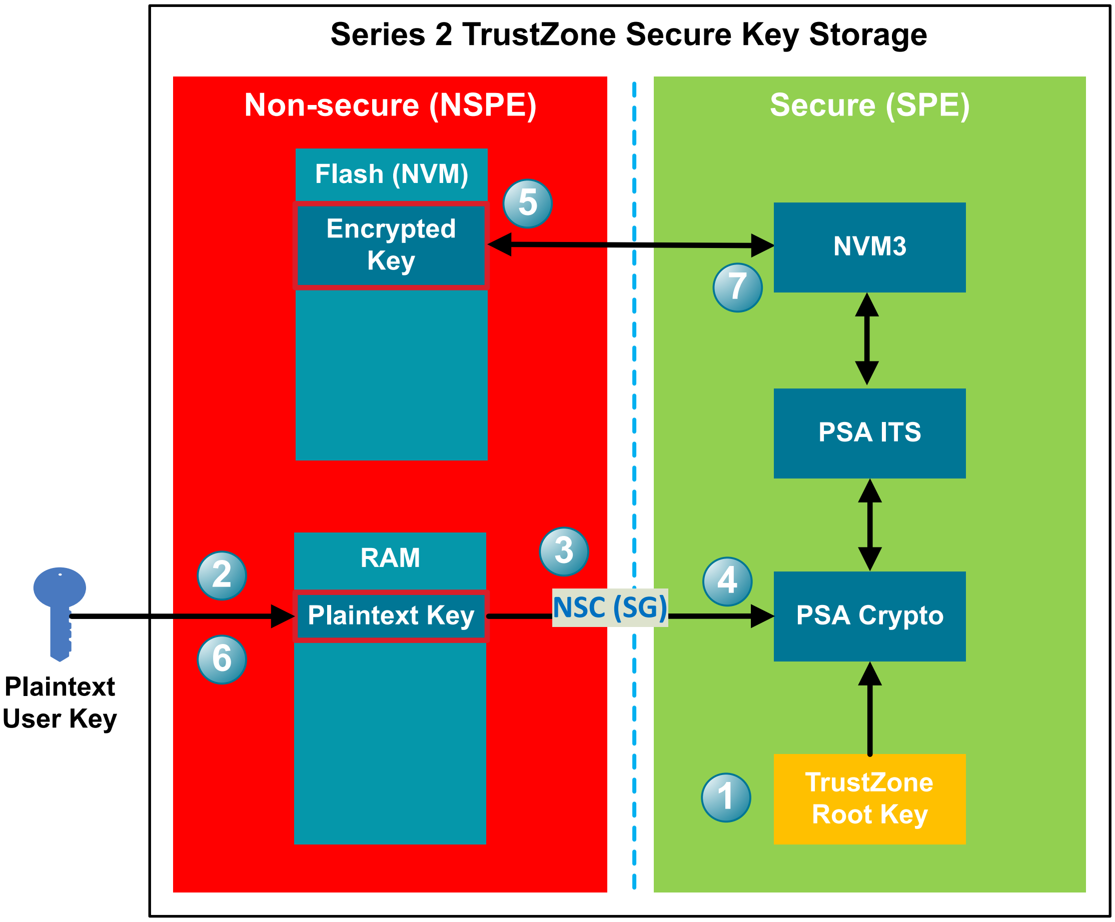
1. The wrapped TRK in VSE NVM is accessed by the VSE Root mode firmware.
2. The PUF-derived key (xG27) or padded unique device serial number (xG22) is used to decrypt (AES-GCM) the wrapped TRK in VSE NVM.
3. Unwrapped TRK is transferred to the shared RAM when switching from VSE Root mode to User mode. The VSE user mode Secure application stores this key to the Secure RAM in SPE and deletes this key in the shared RAM.
4. The unwrapped TRK in the Secure RAM is the master key of a Key Derivation Function (KDF).
5. The encryption key in SPE for Secure Key Storage is derived from the KDF CMAC.

Note: On VSE devices, all cryptographic operations are performed by the Cryptographic Accelerator (CRYPTOACC) peripheral.

For more information about the HSE and VSE, refer to the "Secure Engine Subsystem" section in AN1190: Series 2 Secure Debug.

4.4 TrustZone Secure Key Storage (HSE and VSE)

The TRK allows a user to securely store a key in the Non-secure flash, limiting the number of keys that can be saved only by the amount of Non-secure storage. The following figure describes using the TRK to encrypt a plaintext key and store it in Non-secure NVM.



1. After power-on, the device's TRK (wrapped in HSE NVM and unwrapped in VSE Secure RAM) is available for the SPE.
2. A user key is generated and imported into the device's Non-secure memory. In this example, the key is imported into Non-secure RAM for easy deletion, and the key is lost if device power is removed.
3. Call the PSA Crypto API (`psa_import_key()` or `psa_generate_key()`) through the Secure Gateway (SG) in Non-secure Callable (NSC) memory to generate a key for crypto operations.
4. The plaintext key is passed in the PSA Crypto API to the SPE, where it is encrypted (AES-GCM) with the encryption key derived (KDF CMAC) from the TRK.
5. The encrypted key is stored to the NVM region in the NSPE through the PSA Internal Trusted Storage (ITS) and NVM3 drivers.
6. The plaintext key can now be deleted from the Non-secure RAM.
7. Only the PSA Crypto API in the SPE can retrieve the encrypted key from NVM in the NSPE and decrypt it for crypto operations in the SPE.

Note: Ignore steps 2 and 6 if the plaintext key is randomly generated by the PSA Crypto.

5. Secure Key Storage Implementations

Users can use Secure Engine Manager (SE Manager) or PSA Crypto in the following figure to access the secure key storage on HSE-SVH devices. SE Manager APIs for secure key storage and crypto are usually not considered external APIs. PSA Crypto API abstracts the entropy sources, crypto primitives, and even advanced security features like secure key storage from the calling functions.

Silicon Labs recommends using PSA Crypto API for secure key storage and cryptography whenever possible. It makes the solution more portable and hardware agnostic. In some cases, however, setting up tamper and initializing the secure boot can only be implemented by the SE Manager APIs.

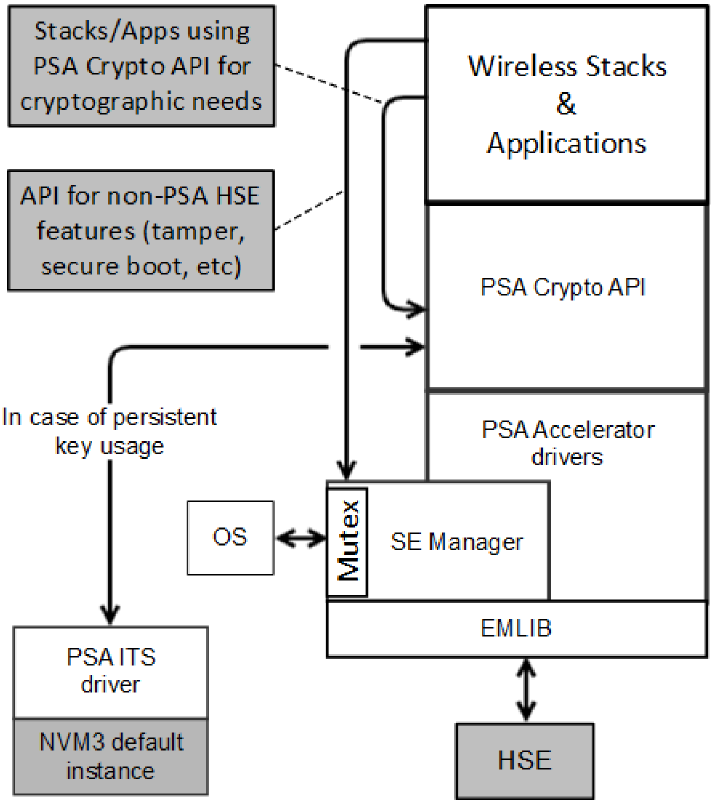

Stacks/Apps using PSA Crypto API for cryptographic needs
API for non-PSA HSE features (tamper, secure boot, etc)
Wireless Stacks & Applications
PSA Crypto API
In case of persistent key usage
PSA Accelerator drivers
OS
Mutex
SE Manager
EMLIB
PSA ITS driver
NVM3 default instance
HSE

Figure 5.1. Secure Engine Manager and PSA Crypto

Table 5.1. Component Description

Component	Functionality
EMLIB (em_se.c)	Abstracts the mailbox interface: how to construct, send and receive low-level HSE mailbox commands.
SE Manager	On top of EMLIB, it abstracts the HSE command set: translates function calls into mailbox messages. The SE Manager also provides thread synchronization.
PSA Accelerator Drivers	A translation layer to map the PSA Crypto HSE interface and crypto acceleration calls to SE Manager calls.
PSA Crypto API	Platform independent cryptographic hardware acceleration support by implementing standardized APIs.
PSA ITS Driver	The key management functionality in PSA Crypto needs access to non-volatile memory for persistent storage of plaintext or wrapped keys. NVM3 gets wrapped by this translation layer, mapping the PSA ITS (Internal Trusted Storage) interface to NVM3 calls.

For the SE's mailbox interface, see section "*Secure Engine Subsystem*" in AN1190: Series 2 Secure Debug.

For more information about NVM3, see https://docs.silabs.com/gecko-platform/latest/driver/api/group-nvm3.

For more information about PSA Crypto, see AN1311: Integrating Crypto Functionality Using PSA Crypto Compared to Mbed TLS.

5.1 SE Manager API

The following table lists the SE Manager APIs related to Secure Key Storage operations. The SE Manager API document can be found at https://docs.silabs.com/gecko-platform/latest/service/api/group-sl-se-manager.

Table 5.2. SE Manager API on Secure Key Storage

SE Manager API	Usage
sl_se_generate_key	Generate a new key and store it either in a volatile HSE storage slot or as a wrapped key.
sl_se_import_key	Import a plaintext key and store it either in a volatile HSE storage slot or as a wrapped key.
sl_se_export_key	Export a volatile or wrapped key back to plaintext if allowed. It will fail for a key that has been flagged as SL_SE_KEY_FLAG_NON_EXPORTABLE.
sl_se_transfer_key	Transfer a volatile or wrapped key to another storage option (volatile HSE storage slot or a wrapped key) if allowed.
sl_se_delete_key	Delete a key from a volatile HSE storage slot.

5.2 PSA Crypto API

The following table lists the PSA Crypto APIs related to Secure Key Storage operations. The PSA Crypto API document can be found at https://docs.silabs.com/mbed-tls/latest/.

For more information about PSA Crypto APIs on Secure Key Storage, see AN1311: Integrating Crypto Functionality Using PSA Crypto Compared to Mbed TLS.

Table 5.3. PSA Crypto API on Secure Key Storage

PSA Crypto API	Usage
psa_generate_key	Generate a new plaintext or wrapped key and store it either in volatile or non-volatile memory.
psa_import_key	Import a plaintext key and save it in plaintext or wrapped form. It can store either in volatile or non-volatile memory.
psa_export_key	Export a key back to plaintext if allowed. The policy on the key must have the usage flag PSA_KEY_USAGE_EXPORT set.
psa_copy_key	Copy key material from one location to another, which may have a different lifetime (e.g., volatile to non-volatile).
psa_destroy_key	Destroy a key from both volatile memory and, if applicable, non-volatile storage.

5.3 SE Manager API Versus PSA Crypto API

The following table compares the SE Manager APIs with PSA Crypto APIs on Secure Key Storage.

Table 5.4. SE Manager API Versus PSA Crypto API on Secure Key Storage

Item	SE Manager API	PSA Crypto API
Availability	Only on HSE devices	Platform independent
API	Silicon Labs proprietary	Standardized by ARM®
Key Storage	Volatile (RAM) memory only	Volatile (RAM) or non-volatile (flash) memory
Wrapped Key Cache	Can use a volatile HSE storage slot	Not yet implemented
Password Protection	Can define in a key descriptor	Not yet defined in PSA Crypto
Custom ECC Curve	Can define in a key descriptor	Not yet defined in PSA Crypto

5.4 PSA Crypto Key Types with TrustZone Secure Key Storage

The following table describes the storage differences between key storage with and without TrustZone on SVM and SVH devices.

Table 5.5. TrustZone Secure Key Storage (SKS) on SVM Devices

Key Type	Storage without TrustZone SKS	Storage with TrustZone SKS
Volatile Plaintext	RAM	Secure RAM (2)
Persistent Plaintext	NVM	Encrypted in NS NVM (2)
Volatile Wrapped	Not supported	Not supported
Persistent Wrapped	Not supported	Not supported

Table 5.6. TrustZone Secure Key Storage (SKS) on SVH Devices

Key Type	Storage without TrustZone SKS	Storage with TrustZone SKS
Volatile Plaintext	Plaintext key in RAM	Plaintext key in Secure RAM
Persistent Plaintext	Plaintext key in NVM	Encrypted plaintext key in NS NVM
Volatile Wrapped	Wrapped key in RAM (1)	Wrapped key in Secure RAM
Persistent Wrapped	Wrapped key in NVM (1)	Encrypted wrapped key in NS NVM

Notes:

- The NVM or NS NVM is at the last part of the main flash.
- It is possible to replace the wrapped key solution on the SVH device (1) with TrustZone Secure Key Storage on the SVM device (2), but this is a less secure approach.

6. Examples

Simplicity Studio 5 includes the SE Manager and PSA Crypto platform examples for Secure Key Storage. Refer to the corresponding `readme` file for details about each SE Manager and PSA Crypto platform example. This file also includes the procedures to create the project and run the example.

Table 6.1. Platform Examples for Secure Key Storage

Category	SE Manager Platform Example	PSA Crypto Platform Example
Key Handling	SE Manager Symmetric Key Handling	PSA Crypto Symmetric Key
	SE Manager Asymmetric Key Handling	PSA Crypto Asymmetric Key
Symmetric Key Usage	SE Manager Block Cipher	PSA Crypto AEAD
		PSA Crypto Cipher
		PSA Crypto KDF
		PSA Crypto MAC
Asymmetric Key Usage	SE Manager Digital Signature (ECDSA and EdDSA)	PSA Crypto DSA
	SE Manager Key Agreement (ECDH)	PSA Crypto ECDH
X.509 Certificate	—	PSA Crypto X.509
TrustZone Secure Key Storage	—	tz_psa_crypto_ecdh_ws

7. Revision History

Revision 0.5

June 2023

- Updated table and note in 1. Series 2 Device Security Features.
- Replaced Device Compatibility with SE Firmware in 1. Series 2 Device Security Features.
- Updated front page and 2. Introduction for TrustZone Secure Key Storage.
- Updated and moved ARM® TrustZone® Key Storage to 4. TrustZone Secure Key Storage.
- Moved sections in 3.2 Plaintext Key Storage and 3.3 Secure Key Storage to sub-sections.
- Modified "Exporting a Wrapped Key" to 3.5.2 Access a Wrapped Key.
- Added 5.4 PSA Crypto Key Types with TrustZone Secure Key Storage.
- Updated 6. Examples and removed the sub-sections.

Revision 0.4

March 2022

- Added digit 4 to Note 3 in 1. Series 2 Device Security Features.
- Updated Device Compatibility and moved it under 1. Series 2 Device Security Features.

Revision 0.3

January 2022

- Added UG489 to the table in 1.2 Key Reference.
- Modified the content and added a note in 3.3 Secure Key Storage.
- Fixed a typo in Figure 3.5 Wrapped Key Import on page 10.
- Modified the content in 3.5.1 Root Key Generation.

Revision 0.2

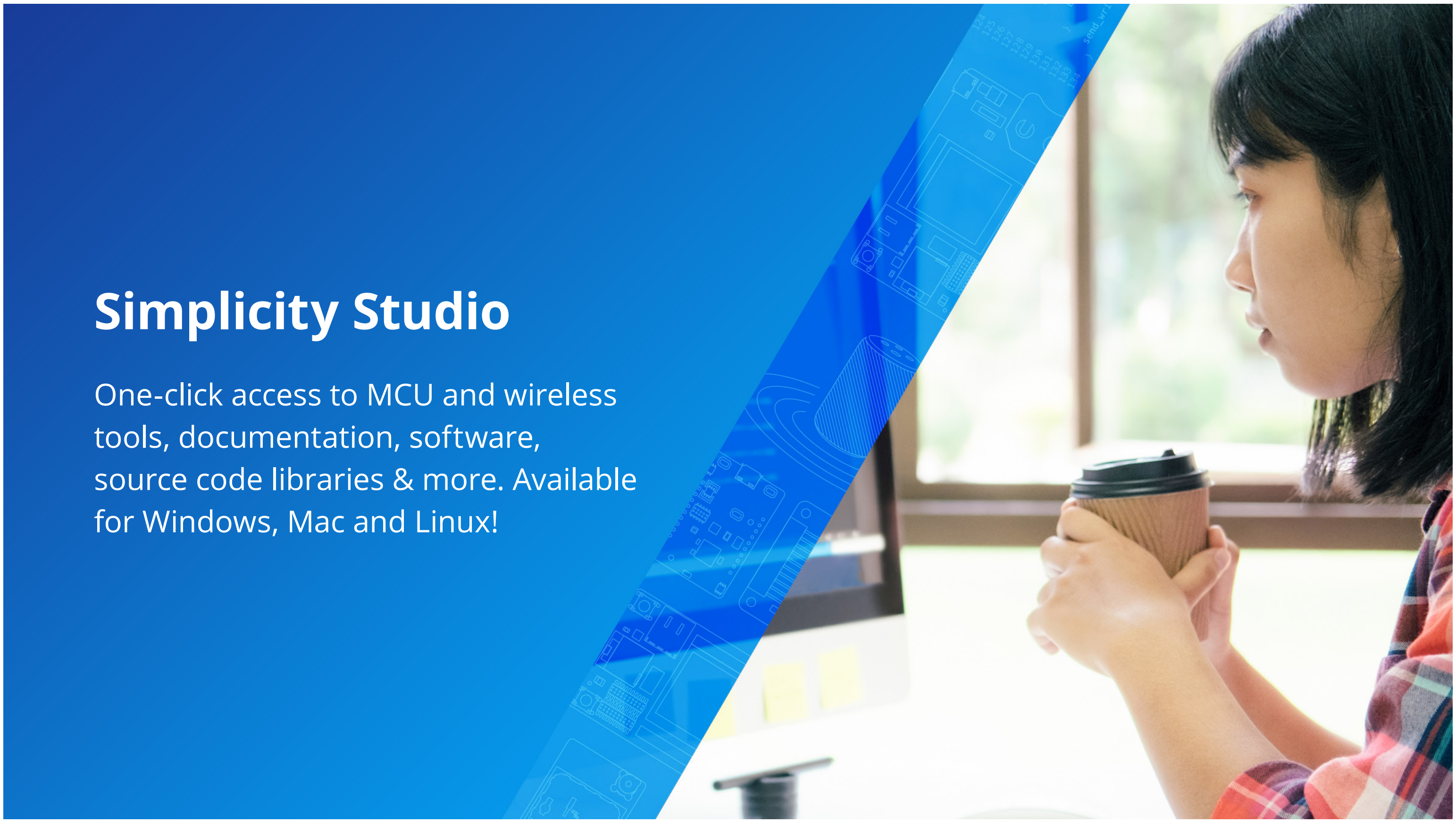
October 2021

- Formatting updates for source compatibility.
- Added revised terminology to 1. Series 2 Device Security Features and use this terminology throughout the document.
- Updated Device Compatibility.
- Deleted Secure Element Manager chapter.
- Added 5. Secure Key Storage Implementations.
- Revised 6. Examples to use SE Manager and PSA Crypto platform examples for Secure Key Storage.

Revision 0.1

September 2020

- Initial Revision.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications. **Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project**

Trademark Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com