AN1329: Using Silicon Labs Secure Vault Features with OpenThread

This application note describes how the secure vault features are leveraged in OpenThread applications. It focuses on specific PSA features and emphasizes how these are integrated into the OpenThread stack.

This document focuses on the updates to secure key storage and crypto modules of OpenThread to leverage Vault features.

KEY POINTS

- Features of Secure Vault devices.
- Key Management in OpenThread
- Crypto Modules in OpenThread
- Integration of PSA in OpenThread

1 Introduction

Google's OpenThread is an open-source implementation of Thread. Google has released OpenThread to make the networking technology used in Google Nest products more broadly available to developers, in order to accelerate the development of products for the connected home and commercial buildings.

With a narrow platform abstraction layer and a small memory footprint, OpenThread is highly portable. It supports both system-on-chip (SoC) and network co-processor (NCP) designs. OpenThread implements all features defined in the Thread 1.1.1 Specification. This specification defines an IPv6-based reliable, secure, and low-power wireless device-to-device communication protocol for home and commercial building applications.

Silicon Labs has enhanced OpenThread to work with Silicon Labs hardware. This source code is available on GitHub and also as a software development kit (SDK) installed with Simplicity Studio 5 (SSv5). The SDK includes a fully tested snapshot of the GitHub source code. It supports a broader range of hardware than does the GitHub version, and includes documentation and example applications not available on GitHub.

Some EFR32 Series 2 products offer additional security options through Secure Vault. Secure Vault is a dedicated security CPU that isolates cryptographic functions and data from the host processor core. Devices with Secure Vault (High) offer the following security features:

- Secure Key Storage: Protects cryptographic keys by "wrapping" or encrypting the keys using a root key known only to the Secure Vault.
- 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.

This guide describes how OpenThread applications leverage Secure Vault features using PSA Crypto APIs.

2 Secure Vault

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

All the security features on Silicon Labs Series 2 devices are implemented using Secure Engine and CRYPTOACC (if available). Types of secure engine implementations in Series 2 devices fall in one of the following categories:

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

The Secure Engine implements and extends all cryptography-related hardware accelerations. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

Table 1.1. Cryptographic Hardware Acceleration Features on Silicon Labs Devices

Level (1)	SE Support	Part (2)
Secure Vault High (SVH)	HSE only (HSE-SVH)	EFR32xG2yB (3)
Secure Vault Mid (SVM)	HSE (HSE-SVM)	EFR32xG2yA (3)
“	VSE	EFM32PG2y, EFR32xG2y (4)
Secure Vault Base (SVB)	N/A	MCU Series 1 and Wireless SoC Series 1

Notes:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. The x can be a letter B, F, M, or Z.
3. At the time of this writing, the y is a digit 1 for the HSE device.
4. At the time of this writing, the y is a digit 2 for the VSE device.

The 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 to be executed.
- 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.

The Secure Vault High offer additional security options as follows:

- 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.

2.1.1 Secure Key Storage

One of the key features of Secure Vault is secure key storage (see AN1271: Secure Key Storage for more information). The cryptographic keys are encrypted by the device’s root key before they are stored in memory for later use. These keys are then available only through the Cryptographic APIs and are referenced by the application with their unique key reference. This allows potentially unlimited secure key storage in any storage location. The keys themselves can be either non-volatile or temporary and the user can choose the key storage location using the attributes.

OpenThread uses the following mechanism to store and use keys available in the stack.

- The user generates the key and specifies the crypto operation that is allowed by the key and the access settings (whether the key can be exported).
- The application passes this key to Secure Vault. Secure Vault then wraps the key and stores it in the specified memory (volatile or non-volatile, based on user settings).
- The application then provides the user with a reference that can be used for Crypto operations.

2.1.2 PSA Crypto

Platform Security Architecture (PSA) is designed by ARM to address the security requirements for IoT devices. It is made up of four stages.

- Threat modelling
- Predefined architectural choices
- Standardized implementation
- Certification

PSA Crypto is one of the standardized implementations of the features recommended by PSA. It implements low-level APIs for cryptographic operations, optimized for MCUs and Wireless SoCs. The PSA Crypto APIs provide easy-to-use interfaces for crypto primitives. These APIs are based on the idea of secure key storage. Access to keys is restricted to the Cryptographic APIs.

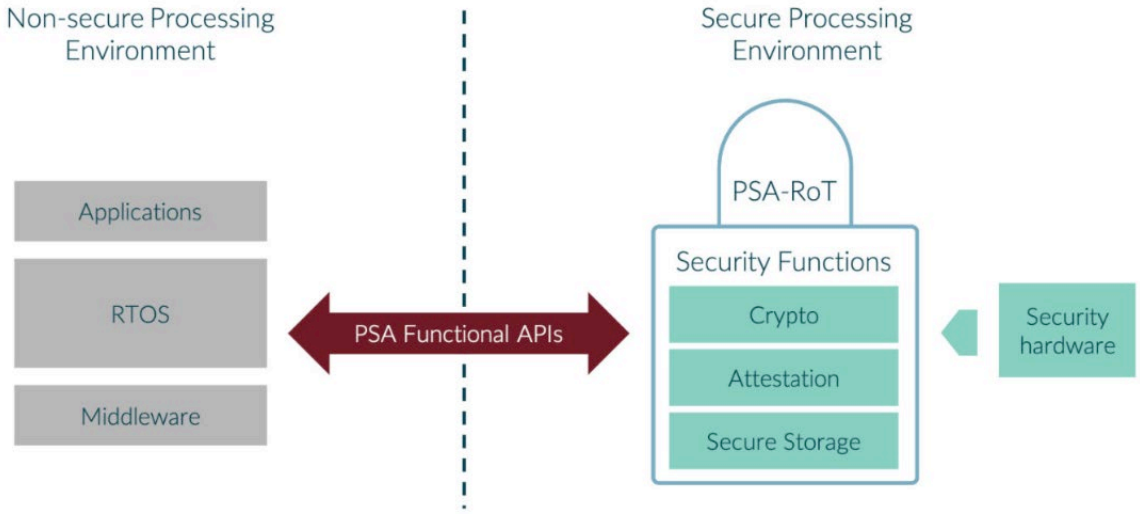

Figure 1.1. PSA Functional APIs

3 Security Handling in OpenThread

Several cryptographic keys are described and used in the Thread protocol. These keys are stored in different modules in the OpenThread stack. All the keys, apart from the Thread Master Key and PSKc, are volatile and are not stored in non-volatile memory (NVM). OpenThread implements a Key Manager module to handle most of the keys required by the Thread protocol.

The following table lists the cryptographic keys used by the OpenThread stack and the module that handles these keys.

Table 2.1. Cryptographic keys in OpenThread stack

Key	Usage	Module	Storage location
Thread Master Key	Used to derive the MAC and MLE keys	Key Manager	NVM
Pre-shared Key for the Commissioner (PSKc)	Used to establish a commissioner session	Key Manager	NVM
MLE Key	Used for Mesh Link Establishment Used by MLE for transmit and receive operations	Key Manager	RAM
Temporary MLE key	Used to process received UDP packet, if the incoming packet has a different keySequence number than that used in the device.	Key Manager	RAM
Key Encryption Key	Used by MAC to process incoming and outgoing packets with keyIdMode0 as 802.15.4 security material	Key Manager	RAM
Pre-shared key for the device (PSKd)	Used in the DTLS exchange to verify the identity of the Joiner	Key Manager	RAM
MAC Previous Key	MAC key for **key sequence = current seq - 1**	MAC	RAM
MAC Current Key	MAC key for **key sequence = current seq**	MAC	RAM
MAC Next Key	MAC key for **key sequence = current seq + 1**	MAC	RAM

OpenThread has a modular approach to security. Each layer of the stack has security implementations and uses common modules for the actual security processing. The current implementation of OpenThread in GitHub uses mbed TLS for all the security operations and, as such, uses all the keys in plaintext for these operations.

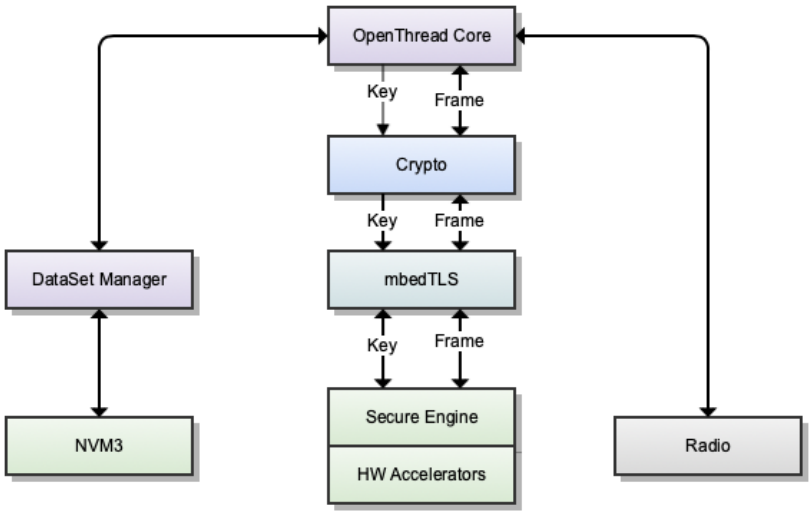

Figure 2.1. Crypto Module Operation in OpenThread

4 Updates to OpenThread to Include PSA Crypto APIs

The OpenThread stack supports crypto operations using either literal keys or key references. This can be configured using the macro **OPENTHREAD_CONFIG_PLATFORM_KEY_REFERENCES_ENABLE**. Also, Openthread stack now supports PSA or mbedTLS APIs for Crypto operations. The stack defaults to using mbedTLS APIs, but it can be configured to use PSA using **OPENTHREAD_CONFIG_CRYPTO_LIB**.

Note: The OpenThread sample apps provided in the SDK have the PSA Crypto with key references enabled by default, except for radio co-processor (RCP) and network co-processor (NCP) applications.

In general, the following points highlight the general changes in OpenThread stack to integrate PSA Crypto API usage.

- All the keys stored in the `key_manager` and `sub_mac` modules are replaced with key references.
- When `key_manager` and `sub_mac` receive security keys, they pass the keys to the PSA abstraction for storage, and only retain the references to these keys.
- The packets in any Rx or Tx events are updated with the relevant key references so that PSA Crypto modules can use them to perform security processing.

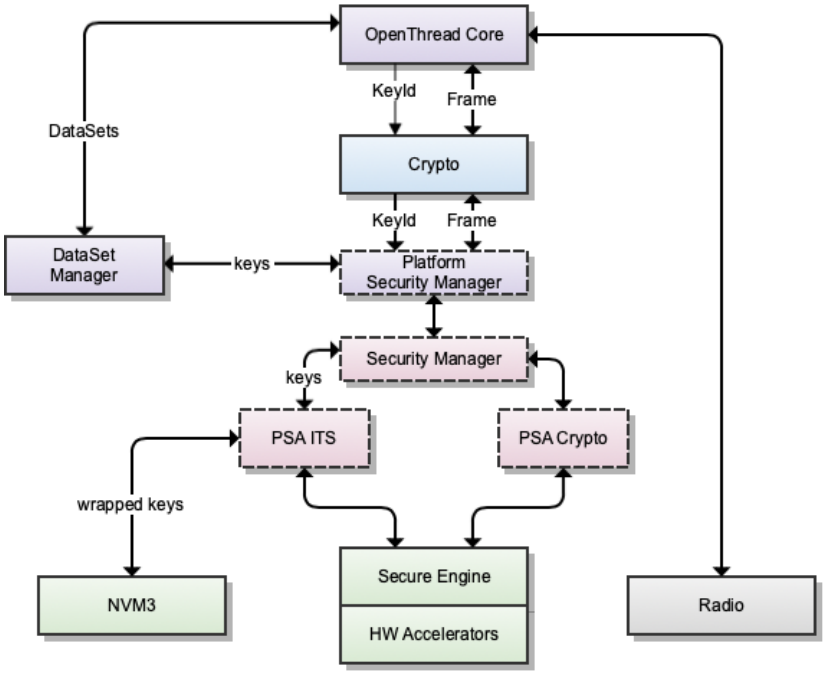

Figure 3.1. Crypto Module Operation in OpenThread with PSA

All keys are wrapped before storing and all the keys are stored in volatile memory (RAM), apart from **Thread Master Key** and **PSKc.** These are stored in NVM to be retained and re-used in the future. The following table lists the security keys stored in Secure Vault along with their corresponding PSA attributes.

Table 3.1. Cryptographic Keys in the OpenThread Stack

Key	Type	Usage	Persistence
Thread Master Key	PSA_KEY_TYPE_HMAC	PSA_KEY_USAGE_SIGN_HASH PSA_KEY_USAGE_EXPORT	PSA_KEY_LIFETIME_PERSISTENT
PSKc	PSA_KEY_TYPE_RAW_DATA	PSA_KEY_USAGE_EXPORT	PSA_KEY_LIFETIME_PERSISTENT
MLE Key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT	PSA_KEY_LIFETIME_ VOLATILE
Temp MLE key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT	PSA_KEY_LIFETIME_ VOLATILE

Key	Type	Usage	Persistence
Key Encryption Key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT PSA_KEY_USAGE_EXPORT	PSA_KEY_LIFETIME_ VOLATILE
MAC Previous Key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT	PSA_KEY_LIFETIME_ VOLATILE
MAC Current Key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT	PSA_KEY_LIFETIME_ VOLATILE
MAC Next Key	PSA_KEY_TYPE_AES	PSA_KEY_USAGE_ENCRYPT PSA_KEY_USAGE_DECRYPT	PSA_KEY_LIFETIME_ VOLATILE

Note: The keys part of the dataset is still stored in the NVM, and is only copied into Secure Vault.

Crypto modules in OpenThread are updated to support the PSA APIs. These APIs use the key references provided by the stack to perform security processing on the incoming/outgoing messages.

4.1 Security Manager

A new platform abstraction has also been introduced to abstract some of the security operations from the core stack. This abstraction acts as an interface to the PSA module and extends PSA interfaces to the stack.

Note: As the PSA does not support the PBKDF2 yet, the stack uses mbed TLS APIs for these operations.

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

SILICON LABS

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

www.silabs.com