

External Trusted Secure Storage Partition

Rev. 1.0, 2022-04-01

Contents

1. Introduction.....	2
2. Design concept	3
2.1 Overview.....	3
2.2 Architecture.....	4
2.2.1 ETSS services.....	4
2.2.2 Secure Flash framework	5
2.3 Code structure	5
2.3.1 Interface	5
2.3.2 etss_partition	6
2.3.3 suites.....	6
3 Hardware and software environment setup.....	6
3.1 Hardware setup	7
3.2 Software setup	7
4 Step-by-step execution.....	7
4.1 Build and load.....	7
4.2 Run test suites	8
5 Revision History.....	9

1. Introduction

This document firstly introduces the motivation of adding External Trusted Secure Storage service(in short, ETSS service), then describes the design proposal and integration guide.

As semiconductor process nodes continue to shrink, the advanced MCUs and SoCs suppliers are shrinking the embedded non-volatile memory footprint. The capacity of embedded NOR Flash is not enough to store sophisticated application code and data. Nowadays more and more application code and data are stored in external memory. Having a secure storage solution is very important when storage is external to MCU. Macronix and other Flash memory suppliers have developed several security memory products, and three major products are RPMC, Authentication Flash, and a more full featured secure Flash like Macronix ArmorFlash.

RPMC is a memory device that its sole purpose is to provide non-volatile monotonic counters for the host. Authentication Flash only performs authentication with the host before operations. Compared to previous two security Flash, the full featured secure Flash performs authentication, encryption along with a full range of additional security features. This secure Flash generally equips with hardware crypto engine with advanced cryptography algorithms, physically unclonable function(PUF), non-volatile monotonic counters, TRNG, key storage and management module, etc.

Secure Flash is always shielded by advanced security functionality, only authorised host is permitted to perform read, write operations. The communication channel between host MCU/SoC and secure Flash is protected by encryption, authentication, data scrambling, and frame sequencing with monotonic counters, as shown in figure 1-1. Besides, the independent secure sections configured with specific security policy satisfies multi-tenant isolation. Hence the secure Flash provides dependable defense against unauthorised access, man-in-the-middle, replay, sniffing and other security threats. s

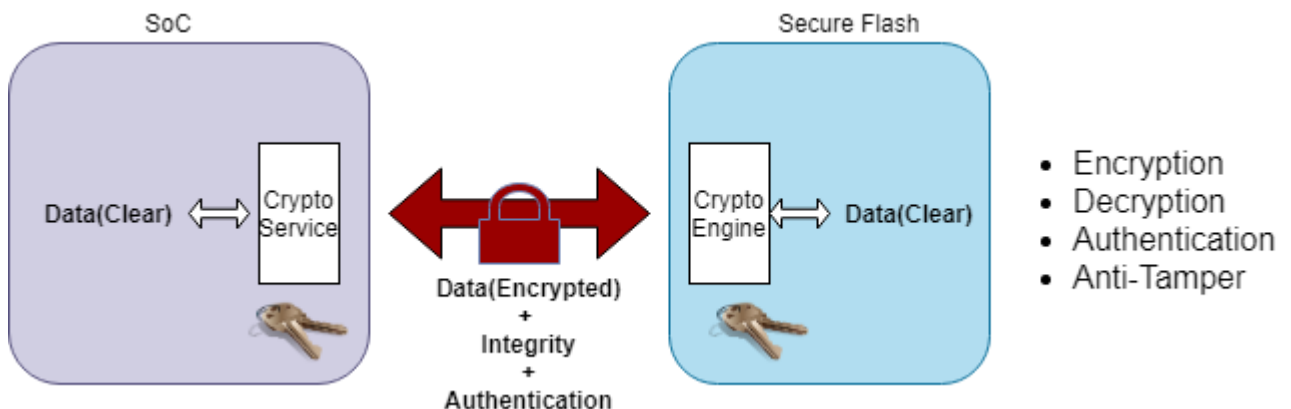


Figure 1-1 Secure communication channel between host and secure Flash

Trusted Firmware-M(TF-M) is a reference implementation of secure world software for Armv8-M, Armv8.1-M architectures. Please refer to TF-M website for more information about TF-M. An External Trusted Secure Storage (ETSS) partition is developed to integrate with native TF-M project, this ETSS partition mainly provides secure external storage services based on secure Flash security features. This integration of secure Flash with TF-M framework will provide an ideal secure external storage solution.

2. Design concept

2.1 Overview

The ETSS partition is developed as a PSA RoT secure partition, it includes several software components, which are listed as table 2-1.

Table 2-1 ETSS partition components

Component name	Description
NSPE client API interface	This module exports the client API of ETSS service to the NSPE (i.e. to the applications).
SPE client API interface	This module exports the client API of ETSS service to the other services available in TF-M.
Service module	This module services the calls from SPE/NSPE client API interface.
Secure Flash framework module	This module is the generic framework of secure Flash driver.

The interaction between these different components is shown as figure 2-1.

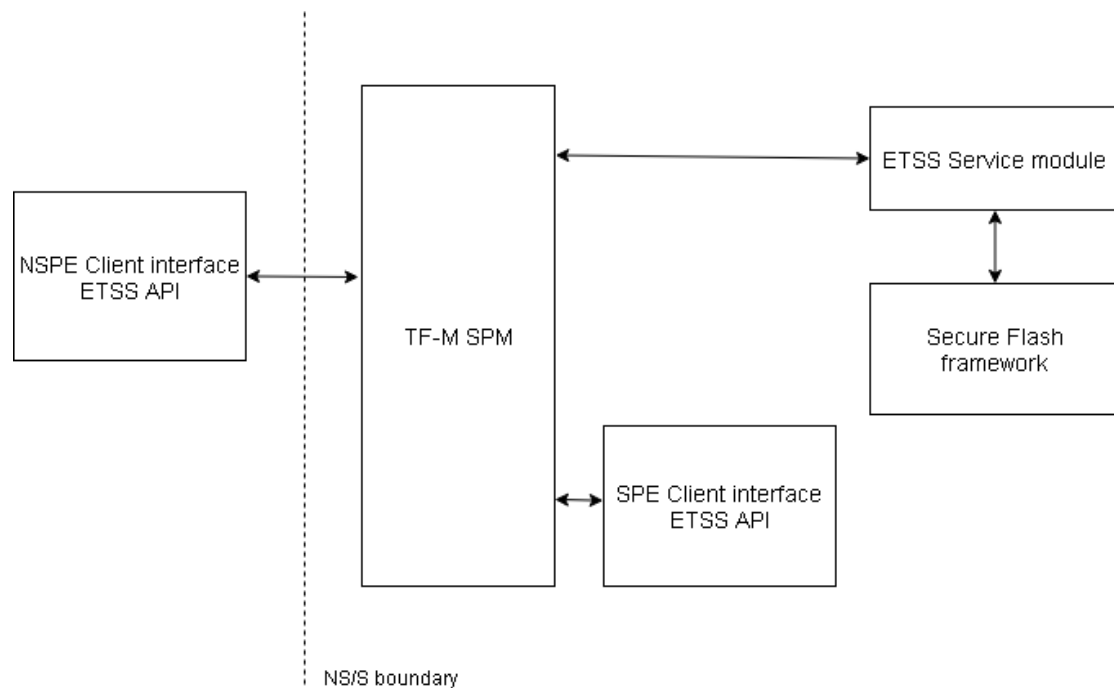


Figure 2-1 Block diagram of the different components of ETSS partition

2.2 Architecture

Figure 2-2 describes the detailed architecture of ETSS services based on secure Flash framework.

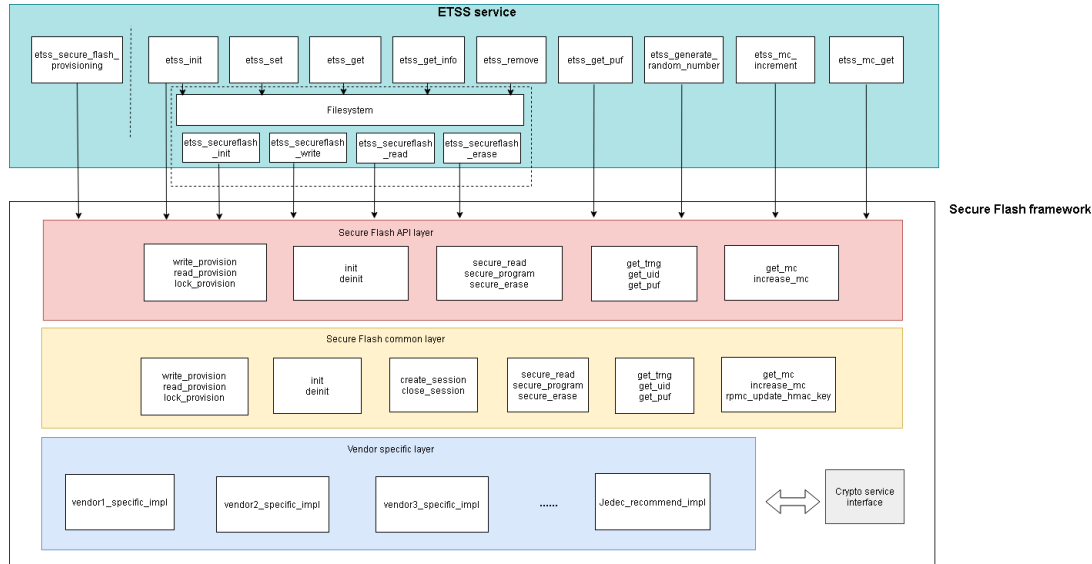


Figure 2-2 Layered architecture of ETSS partition

2.2.1 ETSS services

ETSS services can be used by other services running in the SPE, or by applications running in the NSPE. ETSS services are divided into two functional parts: provisioning and deployment. A provisioning process should be performed to set up binding keys and grant access rights before deployment.

The `etss_secure_flash_provisioning` service aims to derive binding keys, application isolation information and configure secure Flash based on received secure Flash provisioning message. The protocol of provisioning message may vary with different secure Flash products and application scenarios. Users should implement appropriate provisioning flow in a secure environment based on practical device designs.

After provisioning, ETSS is ready for providing deployment services with external secure Flash. These services are mainly classified into three types: secure storage, replay protection monotonic counter manipulation, extra security features (such as PUF, true random number generator, etc.).

The actually available services are based on the security features of backend secure Flash.

Taking following scenarios for example:

- The external security memory product is just an RPMC, then only monotonic counters manipulation services are available.
- The external security memory product is a full featured secure Flash, it supports security read, security program, has a certain number of monotonic counters and also has extra security functions. Then the general ETSS services may be available.

Currently, ETSS shares the simple filesystem of TF-M ITS, because ETSS is also a PSA RoT partition, it calls ITS filesystem APIs directly.

As this simple filesystem doesn't involve access rights management, to support secure Flash multi-zone isolation, the current approach is to declare separate filesystem contexts for each secure Flash isolated partition. And the layout of each isolated partition is configured in `secureflash_layout.h` of each specific secure Flash.

If user needs to support two and more security memory products simultaneously in the ETSS partition, then corresponding secure Flash instances and filesystem contexts should be declared.

2.2.2 Secure Flash framework

The secure Flash framework module aims to generalize the application interface of secure Flash driver, and meanwhile be compatible with different vendors' security memory products. It can be integrated with different software platforms and OSes. Currently, this framework mainly consists of four parts: secure Flash API layer, secure Flash common layer, vendor specific layer and crypto service interface.

Secure Flash API layer: This layer is the interface to upper layer, it mainly manages application's access permission based on application identification and pre-provisioned information. The implementation of this layer varies across software platforms and OSes.

Here integrated with TF-M, this layer manages access permissions based on client id, and derives corresponding access parameters.

Secure Flash common layer: This layer could be understood as an abstraction layer on top of more concrete vendor specific operations.

Vendor specific layer: The specific implementation of different secure Flash vendors and JEDEC recommended implementation, it depends on application's choice to bind with JEDEC recommended implementation or to bind with vendor specific implementation.

This layer calls `tf-m` crypto services via crypto service interface to perform cryptographic operations, then assemble packets sent to external secure Flash and parse packets received from external secure Flash. For each specific security memory products, the corresponding `secureflash_layout.h` should be edited according to application scenarios.

Given that security memory vendors tend to release with hiding some critical source codes, which means these critical parts maybe released as library files. User should also achieve these library files and put them in the appropriate locations.

2.3 Code structure

This package consists of three folders:

2.3.1 Interface

This folder holds the interface for NSPE.

- ``include/etss/etss_api.h`` - ETSS API
- ``include/etss/etss_defs.h`` - ETSS definitions
- ``src/etss/etss_ipc_api.c`` - ETSS API implementation for NSPE

2.3.2 etss_partition

This folder holds the etss partition implementation.

- ``etss.yaml`` - ETSS partition manifest file
- ``etss_secure_api.c`` - ETSS API implementation for SPE
- ``etss_req_mgr.c`` - Uniform IPC request handlers
- ``external_trusted_secure_storage.h`` - ETSS API with client_id parameter
- ``external_trusted_secure_storage.c`` - ETSS implementation, using the flash_fs as a backend
- ``external_secure_flash/`` - Secure Flash filesystem operations
- ``secureflash/`` - The backend secure Flash framework for ETSS service
 - ``secureflash.c`` - The secure Flash API layer interfaces implementation
 - ``secureflash.h`` - The secure Flash API layer interfaces
 - ``secureflash_common/`` - The secure Flash common layer of secure Flash framework
 - ``crypto_interface/`` - The crypto service interface of secure Flash framework
 - ``JEDEC_recommend_impl/`` - The reserved JEDEC recommend uniform implementation
 - ``macronix/`` - Macronix specific implementation
 - ``secureflash_vendor2/`` - The reserved secure Flash vendor2 specific implementation
 - ``secureflash_vendor3/`` - The reserved secure Flash vendor3 specific implementation
- ``template/`` - The templates of hardware platform specific implementation

2.3.3 suites

This folder holds the test suites of etss partition.

- ``suites/etss``
 - ``non_secure/etss_ns_interface_testsuite.c`` - ETSS non-secure client interface test suite
 - ``secure/etss_s_interface_testsuite.c`` - ETSS secure client interface test suite
 - ``secure/etss_s_reliability_testsuite.c`` - ETSS secure interface reliability test suite

3 Hardware and software environment setup

The ETSS partition has been tested with following hardware and software environment.

3.1 Hardware setup

- Macronix MX75 ArmorFlash
- STM32L562E-DK Discovery kit

3.2 Software setup

- Ubuntu
- Python3.6
- gcc-arm-none-eabi-9-2020-q2-update

4 Step-by-step execution

As described before, secure Flash provisioning procedure should be implemented before testing ETSS deployment services.

A simplified secure Flash provisioning template has been contained in *etss_ns_interface_testsuite.c*, user needs to fill the *provision_data* array with their own provisioning data blob. User can also replace this secure Flash provisioning template with their own provisioning implementations.

Besides, user should fulfill their specific implementation of the APIs of *plat_secure_flash.h* and *Driver_SPI.h*. (Assume that the interface between host and secure Flash is SPI interface, otherwise, the other interface driver should be implemented.)

User can also contact Macronix to get assistance.

4.1 Build and load

The following steps describes how to test ETSS partition with existing TF-M framework.

1. Download TF-M v1.4.0 and dependency projects.
2. Download tf-m-extras from <https://git.trustedfirmware.org/TF-M/tf-m-extras.git> to local.
3. Put ``external_trusted_secure_storage/etss_partition`` and
``external_trusted_secure_storage/etss_manifest_list.yaml`` under ``tf-m-extras/partitions``
4. Put ``external_trusted_secure_storage/interface/include/etss`` under ``trusted-firmware-m/interface/include``, put ``external_trusted_secure_storage/interface/src/etss`` under ``trusted-firmware-m/interface/src``, put the ``suites/etss`` folder under ``tf-m-test/test/suites``
5. Add following command line to ``tf-m-test/test/suites/CMakeLists.txt``.
add_subdirectory(suites/etss)
6. Add following command line to ``tf-m-test/app/CMakeLists.txt``
\$($\$$ <BOOL:\${TFM_PARTITION_EXTERNAL_TRUSTED_SECURE_STORAGE}>):\${INTERFACE_SRC_DIR}/etss/etss_ipc_api.c>

7. Build with the following commands:

```
cd <TF-M base folder>
mkdir cmake_build
cd cmake_build
cmake .. -DTFM_PLATFORM=<platform>
-DTFM_TOOLCHAIN_FILE=./toolchain_GNUARM.cmake
-DTEST_NS=ON -DTEST_S=OFF -DTFM_PSA_API=ON
-DTFM_EXTRA_MANIFEST_LIST_FILES=<tf-m-extras-abs-
path>/partitions/external_trusted_secure_storage/etss_manifest_list.yaml
-DTFM_EXTRA_PARTITION_PATHS=<tf-m-extras-abs-
path>/partitions/external_trusted_secure_storage
-DTFM_EXTRA_CONFIG_PATH=<tf-m-extras-abs-
path>/partitions/external_trusted_secure_storage/etss_partition/etss_config.cmake
-DTFM_ISOLATION_LEVEL=2 -G"Unix Makefiles" -
DTFM_PARTITION_EXTERNAL_TRUSTED_SECURE_STORAGE=ON
make install
```

Note:

<platform>: Such as "stm/stm32l562e_dk", user should modify it to actual hardware platform.

<TF-M base folder>: The absolute path of trusted-firmware-m folder.

<tf-m-extras-abs-path>: The absolute path of tf-m-extras folder.

8. Sign the tfm_s and tfm_ns images.

9. Load bl2_s image, tfm_s and tfm_ns signed images to run tf-m-tests.

4.2 Run test suites

After completing the above procedure, you should see the following messages in your serial console.

```
[INF] Starting bootloader
[INF] Swap type: none
[INF] Swap type: none
[INF] Bootloader chainload address offset: 0x19000
[INF] Jumping to the first image slot
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000002
Booting TFM v1.4.0
Non-Secure system starting...

#### Execute test suites for the Non-secure area ####
Running Test Suite external trusted secure storage NS interface tests (TFM_ETSS_TEST_1XXX)...
> Executing 'TFM_ETSS_TEST_1001'
  Description: 'Set interface'
  TEST: TFM_ETSS_TEST_1001 - PASSED!
> Executing 'TFM_ETSS_TEST_1002'
....
```

5 Revision History

Date	Version	Changes
1/4/2022	1.0	First Release.

Except for customized products which have been expressly identified in the applicable agreement, Macronix's products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and not for use in any applications which may, directly or indirectly, cause death, personal injury, or severe property damages. In the event Macronix products are used in contradicted to their target usage above, the buyer shall take any and all actions to ensure said Macronix's product qualified for its actual use in accordance with the applicable laws and regulations; and Macronix as well as its suppliers and/or distributors shall be released from any and all liability arisen therefrom.

Copyright© Macronix International Co., Ltd. 2022. All rights reserved, including the trademarks and tradename thereof, such as Macronix, MXIC, MXIC Logo, MX Logo, Integrated Solutions Provider, Nbit, Macronix NBit, HybridNVM, HybridFlash, HybridXFlash, XtraROM, KH Logo, BE-SONOS, KSMC, Kingtech, MXSMIO, Macronix vEE, Macronix MAP, RichBook, Rich TV, OctaRAM, OctaBus, OctaFlash, and FitCAM. The names and brands of third party referred thereto (if any) are for identification purposes only.

For the contact and order information, please visit Macronix's Web site at: <http://www.macronix.com>.