

---

# **Samsung TEEGRIS Security Architecture**

---

Author	m.sec.teesdk@samsung.com	Date/Version	TEEGRIS v4.x
Organization	Mobile Security Technology group, Samsung Electronics		

Samsung TEEGRIS	
1	Introduction ..... 4
1.1	Objective ..... 4
1.2	Abbreviations and Acronyms ..... 4
2	TEEGRIS Asset ..... 5
2.1	Hardware ..... 5
2.1.1	TEE SoC ..... 5
2.1.2	TEE hardware ..... 5
2.1.3	TEE Root-of-Trust ..... 5
2.2	Software ..... 6
2.2.1	TEE software ..... 6
2.2.2	TEE asset ..... 6
2.2.3	TA asset ..... 6
3	TEEGRIS Security ..... 7
3.1	Isolation ..... 7
3.1.1	Rich execution environment ..... 7
3.1.2	Application ..... 7
3.2	Access Control ..... 7
3.2.1	TA and permission ..... 7
3.2.2	Policy profiles ..... 8
3.2.3	Access control policy ..... 8
3.3	Crypto and Key Management ..... 9
3.3.1	Crypto function call ..... 9
3.3.2	Supported Crypto functions ..... 9
3.3.3	H/W Crypto ..... 9
3.3.4	Key management ..... 10
3.3.5	Supported Key ..... 10
3.4	Trusted Storage ..... 11
3.4.1	Architecture Diagram ..... 11
3.4.2	Storage Structure ..... 11
3.4.3	Rollback Detection ..... 12
3.5	RPMB ..... 12
3.6	ACSD ..... 13
3.6.1	ACSD Provisioning via OTA ..... 13
3.6.2	ACSD Policy Update via OTA ..... 13
3.7	Hardening ..... 14
3.7.1	ASLR ..... 14
3.7.2	KASLR ..... 15
4	Trusted Application ..... 16
4.1	TA Package ..... 16
4.2	TA Integrity protection and signing ..... 16
4.2.1	Local Signing ..... 16
4.2.2	Server Signing - Samsung TA Developer Only ..... 16
4.3	TA anti rollback protection ..... 17
4.4	TA property (GP, TEEGRIS custom) ..... 17
4.4.1	GP properties ..... 17
4.4.2	Custom properties (Optional) ..... 18
4.5	TA memory ..... 19
4.5.1	Memory allocation ..... 19
4.5.2	Inter-world shared memory ..... 20
5	Security Guideline ..... 21
5.1	Assumptions on TA Development ..... 21
5.2	Persistent Object ..... 21
5.3	[outbuf] in Cryptography ..... 21
5.4	Hardware Cryptography ..... 21

---

Samsung TEEGRIS	
5.5	Secure Object ..... 22
5.6	Obfuscation of TA..... 22
5.7	Stack Protector for TA code ..... 22
5.8	Parameter handling ..... 22
6	GlobalPlatform Certificate..... 23
6.1	Protection profile ..... 23
6.2	Certified TEEGRIS ..... 23

# 1 Introduction

SAMSUNG designed TEEGRIS with security and performance in mind. TEEGRIS presents a trusted execution environment and variant security solution based on TrustZone. The variant hardening features are also implemented on TEEGRIS which make a TEEGRIS more secure. HW crypto module are tightly coupled with TEEGRIS, so you can easily, secure and faster use crypto related functions. In addition, TEEGRIS support SMP and pthread, so you can enjoy entire core performance on AP.

## 1.1 Objective

This document provides the security architecture of TEEGRIS which intended to elaborate security aspects of the operating system itself and trusted application development.

## 1.2 Abbreviations and Acronyms

Terminology / Abbreviations	Description
OTP	One-Time Programmable
PP	Protection Profile
REE	Rich Execution Environment
SoC	System on Chip
TEE	Trusted Execution Environment
AP	Application Processor
ATF	ARM Trusted Firmware
SWd	Secure World
NWd	Normal World
TA	Trusted Application
CA	Client Application
TZ	Trusted Zone
ACSD	Access Control Status Database
MMU	Memory Management Unit

## 2 TEEGRIS Asset

This section presents how TEEGRIS provides the principle of its asset protection.

### 2.1 Hardware

TEEGRIS is presented on top of the following hardware components.

#### 2.1.1 TEE SoC

The basic security of TEEGRIS is built upon the ARM TrustZone enabled SoC. The ARM TrustZone is one of the ARM security extensions of adding a secure state, and switching between a secure world and non-secure world. This security extension guarantees a secure privileged operation.

ARM TrustZone provides basic principles what TEEGRIS aims at:

- Securing memory subsystem by address space controller.
- Securing peripherals (e.g., i2c, display, spi, timer, and crypto) by peripheral protector.
- Allocating secure interrupts
- Switching to secure world, and processing sensitive operations

#### 2.1.2 TEE hardware

- Physical volatile memory: Internal SRAM, DRAM
- Memory protection unit : external memory interface, (Secure) DRAM
- Secure Peripherals: JTAG, PRNG/TRNG, Crypto Engine, Timer
- Peripherals shared with REE: Flash memory, I2C, SPI, display, keypad, and so on

Further, the security of hardware module is as follows:

- TRNG: unpredictable random, sufficient entropy
- JTAG: irreversibly closed on production device, and controlled access with debug authentication key on development device
- Timer: integrity with monotonicity

#### 2.1.3 TEE Root-of-Trust

- Root-of-Trust for secure boot

The integrity and authenticity of the TEEGRIS initialization code, data and firmware itself is verified based on the Root-of-Trust for secure boot. TEEGRIS Root-of-Trust for secure boot is mainly beginning from the signature verification of its software component.

On most of the SoC, Root-of-Trust key for secure boot is tightly coupled with secure OTP with rigorous hardware module. The Root-of-Trust of secure boot will be injected on the SoC as form of cryptographic hash. It will be verifying the integrity and authenticity of forthcoming bootloader, Linux kernel and system images. These SOC dependent keys will be fused into Secure OTP, during Samsung factory. Any additional system won't program this Root-of-Trust key, and must not be programmed twice (*non-modifiable*). The TEEGRIS binary (embedded into bootloader) will be signed with an OEM defined key, paired with the fused Root-of-Trust key. In the secure manner, the signing process will be done.

- Root-of-Trust for cryptographic operation

The storage of key and data handled by TEEGRIS is based on the Root-of-Trust for cryptographic operation, as a form of hardware cryptographic key. In addition, it expands to protect the *integrity and confidentiality* of persistent trusted storage owned by each application.

SOC has a different way to provide the Root-of-Trust for cryptographic operation; meanwhile it is typically blown as a hardware cryptographic key and separately managed by a dedicated hardware cryptographic block.

## 2.2 Software

TEEGRIS software architecture is presented alongside a REE operating system environment.

### 2.2.1 TEE software

The TEE software consists of:

- TEE operating system: secure kernel providing operating system features
- TEE framework libraries: abstraction layer with rich functionalities in TEE (i.e., TEE API)
  - GlobalPlatform APIs
  - Samsung proprietary APIs
- TEE external interfaces
  - Shared memory between TEE and REE
  - TEE communication component with REE (e.g., TEE driver)

More details can be found in [Samsung TEEGRIS Secure OS overview] document.

### 2.2.2 TEE asset

The list of assets belongs to TEE operating is as follows:

	Security Properties			
	confidentiality	integrity	consistency (runtime)	authenticity
TEE code and .ro data	O (optional, depending on SoC)	O	O	O
TEE runtime data (e.g., execution variables, runtime context in volatile memory during execution)	O	-	O	-
TEE persistent data (e.g., TEE configuration data, Trusted applications properties)	O	O	-	O
TEE API Code (e.g. secure libraries)	-	O	O	O

### 2.2.3 TA asset

The list of assets belongs to TA is as follows:

	Security Properties				
	confidentiality	integrity	consistency (runtime)	authenticity	TA binding
TA code and .ro data	O (optional)	O	O	O	-
TA version	-	O	O	-	O
TA persistent storage (key, data)	O	O	-	O	O
TA cryptographic implementation	-	O	O	O	-

### 3 TEEGRIS Security

TEEGRIS Security is based on TrustZone and secure boot which are covered at section 3. TEEGRIS is protected from non-secure software by hardware separating and guaranteed integrity and authenticity by a secure boot.

TEEGRIS provides various advanced security features. Such features are working seamless and protect not only the TEEGRIS code and data, but the user data, code, and resources. This section covers details about how security features are working and implemented on TEEGRIS.

#### 3.1 Isolation

##### 3.1.1 Rich execution environment

Due to nature of the ARM TrustZone, TEEGRIS cannot be accessed from REE side. This is typically high layer of exception by configuring the protection controller. REE-TEE isolation is made possible upon the MMU configuration. Any further peripheral is also configured as secure, no REE access is allowed.

##### 3.1.2 Application

Applications are executed in separated virtual address spaces. MMUs typically divide the virtual address space (the range of addresses used by the processor) into pages. ARM architecture-based application processors implement an MMU defined by ARM's virtual memory system architecture. TEEGRIS mainly uses such list of MMU registers to perform operations with MMU.

From an application point of view it looks like all the memory is available for it. The Application developer may write his app as if it's only app that will be ever executed in system. This basic approach is called memory isolation. Following this approach guarantees that any application error will affect only application's memory resources and so TEEGRIS itself and other applications are safe. Even more, application that breaks memory usage rules (most probably due to internal error) will generate exception, called 'page fault'. TEEGRIS can detect such exceptions and kill bad application, freeing its resources.

#### 3.2 Access Control

##### 3.2.1 TA and permission

TEEGRIS access control restricts access from TAs to specific system assets. Access control can be represented as a relation between subjects (who is accessing some object) and objects (that are accessed by subjects). Relation between them is regulated by permission that effectively define an operation that subject can have over the object.

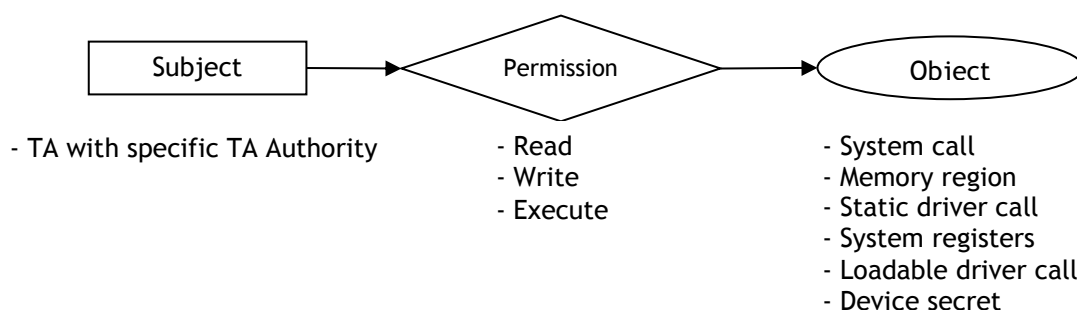


Figure 1 Access Control

### Samsung TEEGRIS

TEEGRIS access control mechanism is used to restrict access from TAs to specific system assets. By system assets here we understand:

- capability to execute specific system calls;
- capability to access to specific memory regions or SoC registers;
- capability to use drivers.

Current implementation of access control mechanism is using pre-defined and hardcoded list of assets

Access control system operates inside of TEEGRIS kernel. No permission checking APIs are exposed to the user space, therefore, TAs cannot change parameters used for access control decision of influence of access control decision result.

### 3.2.2 Policy profiles

Each TA contains special custom TA property indicating its <Policy Profile> as null-terminated string.

The Policy Profile Name is defined by TA Authority and is present in both TA property and a certificate used to verify the TA image integrity. It is the responsibility of the verifier to match the TA Authority Name in certificate and TA property.

Access control mechanism defines the 'policy profile' - a named combination of permissions for each asset. Policy profiles are used to store specific set of permissions for each asset for each TA Authority.

For that name in the Policy Profile Name, the property must match the policy profile name.

Current version of access control mechanism is using several pre-defined policy profiles.

Typical policy profiles for Samsung internal developers are:

- samsung\_drv                      Samsung-owned driver
- samsung\_ta                      Samsung-owned TA

### 3.2.3 Access control policy

The meaning of access bit for each group of assets can be checked with the table below:

Access bit	Groups of assets		
	Drivers	Memory maps	API
X	Open/Execute	Not applicable	Execute
R	Not applicable	mapping memory for RO	Not applicable
W	Not applicable	mapping memory for RW	Not applicable

**Table 1 Asset Groups**

Permissions to the classes of resources will be given to each TA according to the specific policies in Policy Profiles. The TA developers may navigate access control policy in SDK to understand each resource defined in policy profile.

### 3.3 Crypto and Key Management

#### 3.3.1 Crypto function call

Implementation of the TEEGRIS crypto functions are based on GP Internal API specifications. TA developers can use the crypto functions easily via this APIs. These crypto functions needs to be called in order of sequence.

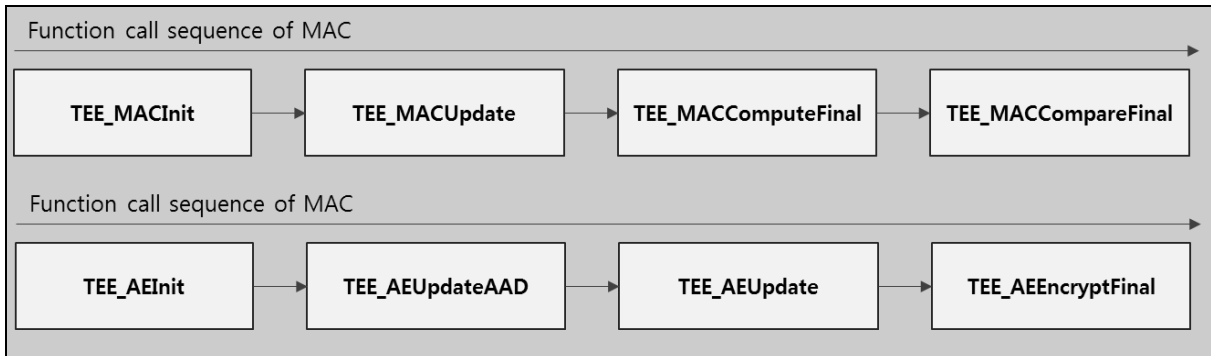


Figure 2 Crypto Function Call

If the wrong function called in this chained sequence, an error will be returned and the crypto functions will stop. It can protect the developer using the wrong crypto function. Also as a security, all crypto functions are operating in the secure world. And internal implementation of crypto functions are using the crypto driver.

#### 3.3.2 Supported Crypto functions

TEEGRIS supports below crypto functions.

- Asymmetric
- AE (Authenticated Encryption)
- Digest
- MAC
- Symmetric
- Generic operation
- KDF (Key Derive Function)
- Random

Currently TEEGRIS provides crypto functions that are based on the GP TEE Internal API 1.1 specification.

In addition, the TEEGRIS TEE does not support DES, and MD5 cryptography algorithm due to security reasons (i.e., end of lifecycle). The application developer should not intend to use such algorithms.

#### 3.3.3 H/W Crypto

Chip vendors provide their H/W crypto features. It calls the hardware cryptographic accelerator. And TEEGRIS use this H/W crypto inside implementation of crypto functions. H/W crypto is faster than general S/W crypto and more safety. Currently RNG, KDF, AES functions support H/W crypto feature. Also in TEEGRIS side, Trusted storage and Secure object using this H/W crypto for security.

Chip vendors provide a unique key which is a “hardware key”. It is used by crypto functions like KDF. This hardware key cannot export to outside of device and each chip has a key of a different unique value. TEEGRIS has a crypto driver as a device driver layer. Inside the crypto driver call H/W crypto codes what provided from Chip vendor. So developer can use the H/W Crypto as well as calling the crypto API of TEEGRIS.

### 3.3.4 Key management

If developer has generated a new key in TEEGRIS, they can manage these keys with its features. TEEGRIS provides a “Secure Object” and “Trusted Storage” for Wrap/Unwrap data. First, secure the object can wrap data as an encrypted blob. This blob is protectively safe because it is encrypted with a unique device hardware key. It means that whoever exports this blob file from a device, the blob will not work on other device. Also this blob has access control functions. So it is possible to set only own TA (secure object created) can access to object or only a delegated TA can access to object. And if the developer uses a trusted storage, it is more easy to store data into the device via Trusted Storage API. But the Trusted Storage use /data/ partition for saving data only. The Secure object side, the developer can store this encrypted blob anywhere.

### 3.3.5 Supported Key

The key of TEEGRIS is used in Crypto functions and it is handled in the form of an object. This object is called the transient object. For example the “TEE\_GenerateKey” function is to generate a new key. But the needed function is the object handler (TEE\_ObjectHandle). This handler includes information about key like key algorithm or key size.

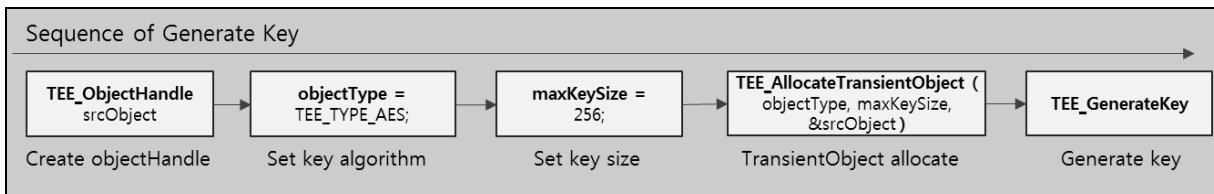


Figure 3 Generate Key Sequence

Supported type list for generate key function is below:

- AES, DES, DES3
- HMAC (MD5, SHA1, SHA224, SHA256, SHA384, SHA512)
- RSA, DSA, ECDSA, ECDH

### 3.4 Trusted Storage

Trusted Storage provides data and key material with guarantees on the confidentiality and integrity of the data stored and atomicity of the operations that modify the storage.

#### 3.4.1 Architecture Diagram

Following diagram represents the architecture.

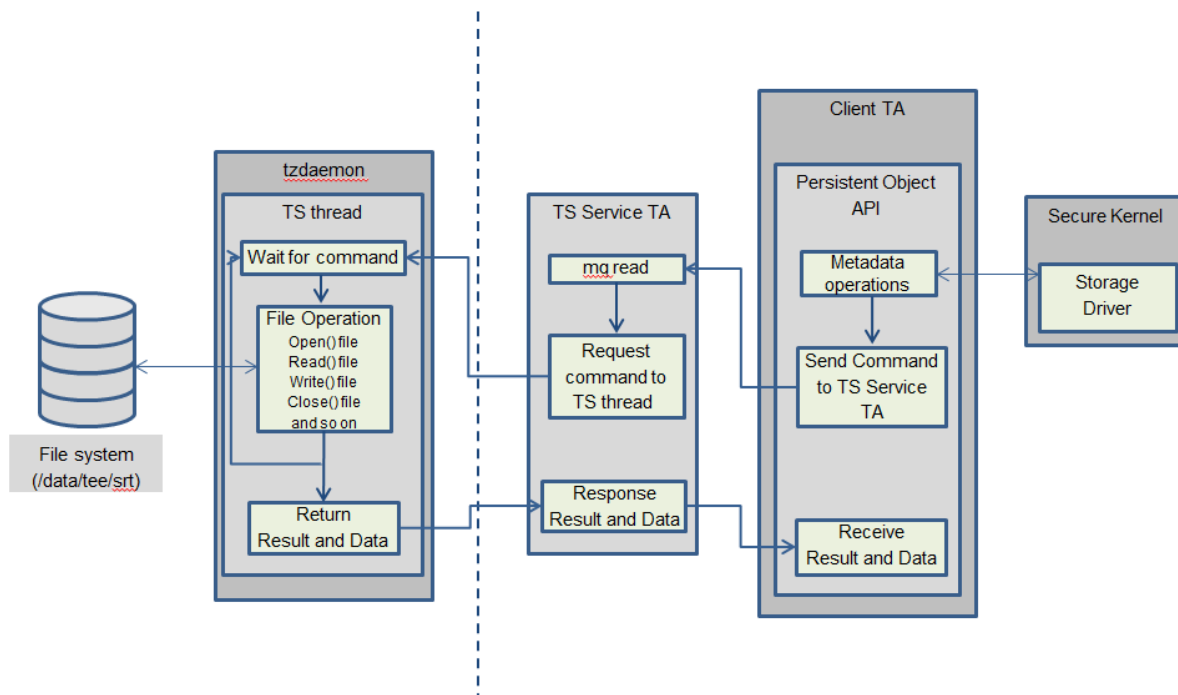


Figure 4 Architecture

#### 3.4.2 Storage Structure

Trusted storage implementation in TEEGRIS makes use of normal world file system to store the data across reboots. While stored in the normal world file system all the content is protected using the Secure Object functionality (AES-GCM based) thus protecting the confidentiality and integrity of the stored data.

Each persistent object can be much larger than the size of total memory available in secure world. Hence in order to handle large objects, object data is split into chunks of an appropriate predefined size.

During the execution of the Trusted Storage API, to load the appropriate file, the data is organized as below.

```

SRT/
| TA-UUID-STORAGE_ID
| TA-UUID
| STORAGE_ID
  
```

## Samsung TEEGRIS

```

| OBJECT_UNIQUE_ID
| OBJECT_INFO
| OBJECT_ATTR (optional)
| DATA_CHUNK (optional)

```

- SRT = Storage Root of Trust (/data/tee/srt)
- TA-UUID-STORAGE\_ID = Contains Storage metadata
- Storage ID = A 32-bit storage identifier. Only Storage ID = 1 is supported for current version.
- Object unique ID = A 32-bit unique Identifier of the Object.
- CHUNK\_INFO = Contains the none data part of the object (Object ID, flags, object version, etc)
- OBJECT\_ATTR = Contains object attributes (key)
- DATA\_CHUNK = Contains the data stream of the object.

### 3.4.3 Rollback Detection

TEEGRIS provides a level of Detection against rollback attacks on persistent Objects

Global Platform allows the level of trust for the rollback Detection to be implementation defined. This can be queried using the property interface on “gpd.tee.trustedStorage.rollbackDetection.protectionLevel”.

TEEGRIS trusted storage supports the protection level 100. This means that the Rollback detection mechanism for the Trusted Storage is enforced at the REE level.

Rollback Detection in TEEGRIS has the version number for each object and for each data chunk of the object. Object version number is stored in object\_info file. Data version number is added to the end of each data chunk and is stored in data\_chunk file. Those numbers are encrypted in TEE and stored in a separate Storage Metadata file.

## 3.5 RPMB

RPMB is special section on EMMC or UFS storage. Only allowed instruction set can access to this RPMB section. TEEGRIS is support RPMB (Replay Protected Memory Block) storage API for store security data. So the developer can use the TA service to use the RPMB storage easily with the RPMB API. The API is simply consists of Read/Write. And RPMB data cannot be resetted or wiped.

RPMB has several partitions; they are based on the chip vendor specifications. Generally each RPMB partition is assigned to each Trusted Applications. So if the TA developer wants to use the RPMB storage, allocation of partition is needed first. Some partitions are allocated for boot loader or section of chip vendor already.

※ As RPMB implementation is dependent upon chipset, TA developer should contact the TEEGRIS administrator.

## 3.6 ACSD

The main purpose of Access Control Status Database is controlling access to system resource from Trusted Applications. To achieve this, the goal authority term was introduced. Each group of TAs (usually belonging to the same 3rd-party developer) has its own authority that defines not only access rights to resources but also allows revoking all TAs belonging to the authority. Additional ACSD instance exists to manage separate TAs to provide fine control over the processes running in the system.

ACSD is responsible for maintenance of white-list based database to allow only authorized TAs to run on the system, installation of 3rd-party TAs and checking their access rights at runtime against rules stored in ACSD.

### 3.6.1 ACSD Provisioning via OTA

ACSD should be securely provisioned at the first system startup and periodically updated to reflect changes to the policy. ACSD provisioning is based on industry standard TLS (Transport Layer Security) protocol for device and server authentication, session key negotiation and protection of message confidentiality and integrity and on Samsung Root Key manager for key pair and device certificates generation. Maintenance of ACSD is performed by a specialized TA responsible for downloading updated policy from OTA server and installing it into the system, a special system TA of TEEGRIS responsible for business logic implementation.

ACSD provisioning is performed at the first device startup. The aim of this process is to acquire the latest ACSD, download TAs initially required for a given device model and to register the device on OTA server to be able to receive updates about ACSD policy. When a device comes off the production line, packaged and shipped, it has not yet been provisioned or enrolled to TEEGRIS OTA server, but it has provisioned SKM service. When the user turns on the device for the first time, the OS typically will ask the user to initialize the system with network connectivity and account information. When TEEGRIS OTA Agent daemon detects that network connection has been established, it starts provision process. First SKM is called to generate certificate for OTA Agent TA. Then OTA Agent TA negotiates mutual TLS connection with OTA server using credentials generated by SKM. TLS connection is established from TEEGRIS secure OS using TEE Sockets API so all data leaving secure world is encrypt to ensure secure connection. OTA server creates a record for new device so now this device is considered enrolled and can receive ACSD updates. OTA Agent TA loads ACSD tokens from server and sends them to Root TA. Now after ACSD is up-to-date it's time to download all TAs required for proper operation of the device. TAs are downloaded using normal world OTA Agent daemon that connects to the same OTA server that provided updated ACSD. Connection to OTA server is secured by normal world's OS TLS, OTA server is authenticated by its certificate while the device is authenticated by the server by nonce signed in TrustZone by OTA Agent TA. After Provisioning stage is complete OTA subsystem is ready.

### 3.6.2 ACSD Policy Update via OTA

ACSD should provide possibility of update to reflect changes in access policies of TEEGRIS. Update may be needed when a new 3<sup>rd</sup>-party developer with new authority is granted access to TEEGRIS. An opposite case is possible when authority already present on a device is revoked for some reasons. Another use case is when a security breach is found on a TA and thus it should be banned from running.

Google Firebase Cloud Messaging is used to deliver updates to each device provisioned on OTA server. The message contains digitally signed token with changes to be applied on ACSD. The Firebase message is received by OTA Java service and sent to Secure World Secure World checks signature of the token to assure its authenticity and applies changes to ACSD. From this moment new policies become effective. No TEEGRIS or device reboot is required.

## 3.7 Hardening

Software can contain bugs which can be exploited to undermine the security. To make these attacks harder, modern operating systems use various methods of hardening. Hardening reduces its surface of vulnerability.

TEEGRIS also supports various hardening solutions for maximum security. This section shows how TEEGRIS provides the hardening solutions.

### 3.7.1 ASLR

ASLR (Address space layout randomization) is a computer security technique involved in protection from buffer overflow attacks. In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.

Address space randomization hinders some types of security attacks by making it more difficult for an attacker to predict target addresses. For example, attackers trying to execute return-to-libc attacks must locate the code to be executed, while other attackers trying to execute shellcode injected on the stack have to find the stack first. In both cases, the system obscures related memory-addresses from the attackers. These values have to be guessed, and a mistaken guess is not usually recoverable due to the application crashing.

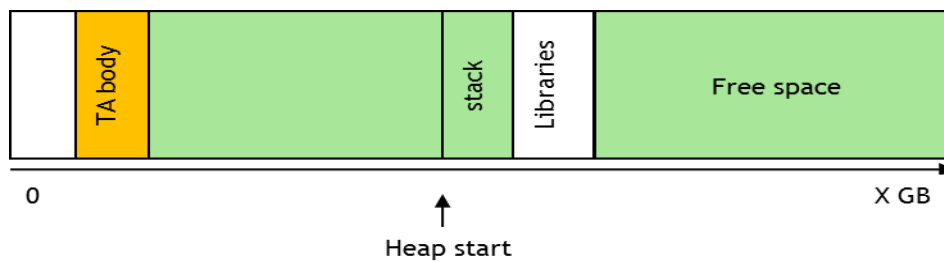


Figure 5 Memory Address Layout without ASLR

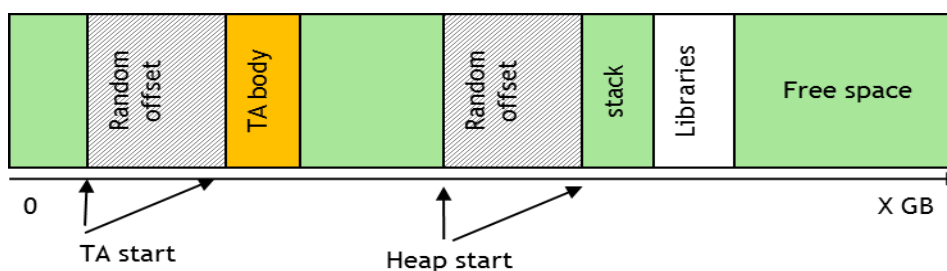


Figure 6 Memory Address Layout with ASLR

TEEGRIS SDK enables ASLR by default and compiles TA code with position-independent executable (PIE) against mentioned attacks. TA developers should carefully understand this mandatory compile option.

### 3.7.2 KASLR

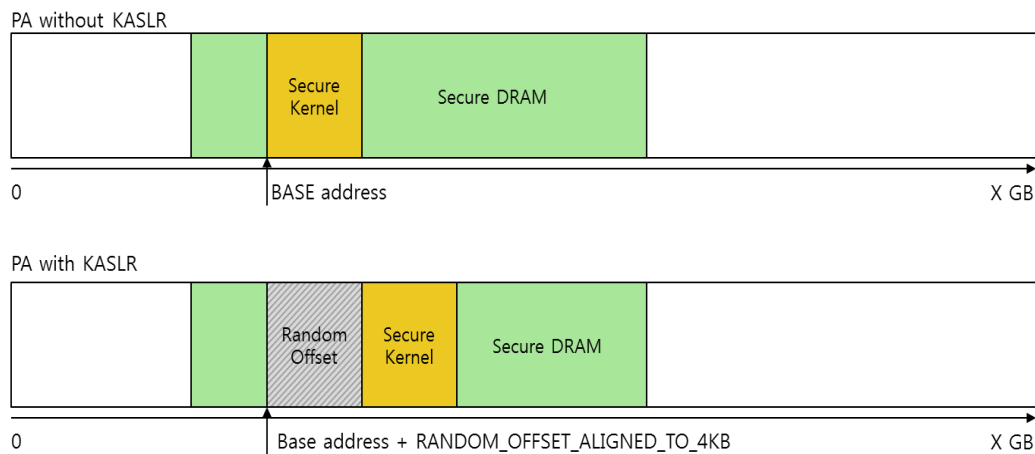
Kernel address-space layout randomization (KASLR) is a technique of placing kernel objects at random addresses in boot time. The main purpose of KASLR is making it difficult to find a starting point of a kernel from the attacker. In TEEGRIS, the work of KASLR occurs before the start\_kernel code in locore. There are two functions of KASLR, first, the physical address randomization, and second, the virtual address randomization. The unit of random address has a page size of 2 MB.

Virtual address randomization was designed with the following steps in mind:

- Find the number of virtual address slots by unit of the huge page size (2 MB)
- Make random numbers (using the True random number generator (TRNG) or Arch time tick)
- Calculate the offset and virtual base address for virtual address KASLR with the number of virtual address slot and random number
- Make and add entry of translation table with KASLR offset and calculated virtual base address

Physical address randomization was designed with the following steps in mind:

- Find the number of physical address slots by unit of the huge page size (2 MB)
- Make random number (use the True random number generator (TRNG) or Arch time tick)
- Calculate the offset for physical address KASLR with the number of physical address slot and random number
- Move kernel image in memory to generated offset



**Figure 7 Physical Address and KASLR**

## 4 Trusted Application

## 4.1 TA Package

TA Package refers to the complete package that is delivered to the device for installation, whereas TA Image refers to the ELF image that will be parsed and loaded.

4 types of package are defined:

- Type 1 - Signed TA package (Deprecated).
- Type 2 - Signed TA package with attached certificate.
- Type 3 - Signed TA package with attached certificate and TA anti-rollback version.
- Type 4 - Signed and Encrypted TA package with attached certificate and TA anti rollback version.

## 4.2 TA Integrity protection and signing

To protect a TA against unauthorized access signing mechanism is used. Every TA binary has to be signed. It was implemented as special script and should be applied to TA binary.

### 4.2.1 Local Signing

Tools for local signing usually are located in TEEGRIS SDK. SDK keeps it in tools/bf\_authority\_scripts/ta\_auth\_scripts. Following is the sample command to sign TA :

- `./sign_file.sh <options> <unsigned_TA> <signed_TA>`

Option	Description
-t <pkg_type>	<p>“SEC1” - Signed TA package (Deprecated).</p> <p>“SEC2” - Signed TA package with attached certificate.</p> <p>“SEC3” - Signed TA package with attached certificate and TA anti rollback version. (-r option is required)</p> <p>“SEC4” - Signed and encrypted TA package with attached certificate and TA anti rollback (-r, -1, -2 option is required)</p>
-c <location>	Define location of Certificate. (default ./ta_auth/cert.pem)
-k <location>	Define location of Private key. (default ./ta_auth/private/key.pem)
-a	Append certificate to signed file (SEC2 type)
-r <version>	Append version information to TA package (SEC3 and SEC4 type)
-e <name>	Use Openssl engine
-p <password>	Define password to unlock the private key
-1 <ta_key>	TA key for encryption
-2 <auth_key>	auth key for encryption
-h	Display usage information

#### 4.2.2 Server Signing - Samsung TA Developer Only

TA on Production model should be signed by the remote signing server. For this purpose the signclient.jar file is required. The file is in the TEEGRIS SDK.

You can get more information from [TEEGRIS User Group](#).

Confidential Property of IT & Mobile Communications, Samsung Electronics Co., Ltd.

### 4.3 TA anti rollback protection

TA Ant-Rollback Protection in TEEGRIS has to have a version number for TA. TA anti-rollback version number is located in the signed TA and integrity of this version is guaranteed by TA signing. Also, the Anti-rollback version is stored in the encrypted database of TEEGRIS. So if the version value of executed TA is less than the stored TA version value, the TA does not work.

If you need to apply TA anti-rollback, you should add the -r option and version value when you sign your TA. Please refer to the Signing command for applying TA Anti-Rollback Protection.

- `./sign_file.sh -t SEC3 -r <version value> <options> <unsigned_TA> <signed_TA>`  
e.) `./sign_file.sh -t SEC3 -r 1 -c cert.pem -k key.pem unsigned_TA signed_TA`

### 4.4 TA property (GP, TEEGRIS custom)

Each TA has a set of mandatory and custom properties that can help TA developers to setup the TA features. Mandatory properties are implemented according to TEE specifications and custom properties are Samsung extensions.

#### 4.4.1 GP properties

Name	Type	Description
TA_PROP_UUID	UUID	16 bits identifier of the Trusted Application.
TA_PROP_SINGLE_INSTANCE	Boolean	Whether the implementation SHALL create a single TA instance for all the client sessions (if true ) or SHALL create a separate instance for each client session (if false ).
TA_PROP_MULTISESSION	Boolean	Whether the Trusted Application instance supports multiple sessions.
TA_PROP_INSTANCE_KEEPAIVE	Boolean	Whether the Trusted Application instance context SHALL be preserved when there are no sessions connected to the instance. The instance context is defined as all writable data within the memory space of the Trusted Application instance, including the instance heap. This property is meaningful only when the TA_PROP_SINGLE_INSTANCE is set to true. When this property is set to false, then the TA instance MUST be created when one or more sessions are opened on the TA and it MUST be destroyed when there are no more sessions opened on the instance. When this property is set to true, then the TA instance is terminated only when the TEE shuts down, which includes when the device goes through a system-wide global power cycle. Note that the TEE MUST NOT shut down whenever the REE does not shut down and keeps a restorable state, including when it goes through transitions into lower power states (hibernation, suspend, etc.). The exact moment when a keep-alive single instance is created is implementation-defined but it MUST be no later than the first session opening.
TA_PROP_DATASIZE	Integer	Maximum estimated amount of dynamic data in bytes configured for the Trusted Application. The memory blocks allocated through TEE_Malloc are drawn from this space, as well as the task stacks. How this value precisely relates to the exact number and sizes of blocks that can be allocated is implementation-dependent.

**Samsung TEEGRIS**

TA_PROP_STACKSIZE	Integer	Maximum stack size in bytes available to any task in the Trusted Application at any point in time. This corresponds to the stack size used by the TA code itself and does not include stack space possibly used by the Trusted Core Framework. For example, if this property is set to “ 512 ”, then the Framework MUST guarantee that, at any time, the TA code can consume up to 512 bytes of stack and still be able to call any functions in the API.
TA_PROP_GROUP_ID	String	Group identifier to which TA belongs to.
TA_PROP_VERSION	String	Version number of this Trusted Application.

#### 4.4.2 Custom properties (Optional)

- GPD.TA.DBG\_DLM.DATA\_AVAILABLE : Debug Rules Property states what sort of DLM data can be retrieved according to the TA.

Possible values		
Name	Value	Description
TEE_BLOCKED	-64	As a TA rule value, indicates that while the TA with this value has active sessions, DLM events of any TA in the TEE shall be ignored and not reported through the DLM service. As a TEE rule value, shall be considered the same as BLOCKED.
BLOCKED	0	No DLM is available. As a TA rule value, indicates that while this TA may open a DLM session and use TEE_DLMPrintf(), there shall be no output. As a TEE rule value, indicates that while any TA may open a DLM session and use TEE_DLMPrintf(), there shall be no output.
ALL	64	All DLM data is available.

- GPD.TA.DBG\_PMR.DATA\_AVAILABLE : Debug Rules Property states what sort of PMR data shall be retrievable, according to the TA

Possible values		
Name	Value	Description
TEE_BLOCKED	-64	As a TA rule value, indicates that while the TA with this value has active sessions, PMR events of any TA in the TEE shall be ignored and not reported through the PMR service.
BLOCKED (default)	0	As a TA rule value, indicates that PMR events of this TA shall be ignored and not reported through the PMR service.
CODE_ONLY	32	Only the following data about the panicked TA shall be available: specNumber functionName markValue panicReasonCode
CODE_STATE	64	This adds the panicked TA's state to the CODE_ONLY set of data available. If sufficient resources are available, the state data shall be placed at the location defined by PMR_STATE_POINTER
HEAP_STACK	96	This adds the panicked TA's heap and stack to the CODE_STATE set of data available. If sufficient resources are available, the heap and stack data shall be placed at the locations defined by PMR_HEAP_POINTER and PMR_STACK_POINTER
ALL	112	All data in the TEE is available to be reported. In systems where TA specific state could not be presented due to the method of sharing TA stacks and heap between trusted applications, in this state such data may be presented.

- SAMSUNG.TA.CACHEHEAPSIZE : Memory area that will be allocated while TA starting. This memory is not exempted with free(). Default value is 0.
- GPD.TA.DESCRPTION : Trusted Application description. Default value is "descr. none ".
- SAMSUNG.TA.FIPS\_MODE\_ENABLE : Enable using FIPS. Default value is false.
- SAMSUNG.TA.NUMINSTANCES : Number of instances. It effects if TA\_PROP\_SINGLE\_INSTANCE = false. Default value is 0.
- SAMSUNG.TA.NUMSESSIONS : Number of sessions. It effects if TA\_PROP\_MULTISESSION = true.
- SYS.TA.NO\_ASLR : Allocation to special memory addresses. Default value is false.
- SYS.TA.RLIM\_CUR : Priority of TA start. Default value is sys.ta.rlim\_max / 2.
- SYS.TA.RLIM\_MAX : Maximum priority value. It is constant.
- SYS.TA.THREAD\_COUNT : Number of threads, default value is 2.

## 4.5 TA memory

### 4.5.1 Memory allocation

TA cannot access and use physical memory directly. And it works in a separated virtual memory space. TEEGRIS supports two types of memory allocator for TA.

- POSIX - malloc, calloc, free
- GP API - TEE\_Malloc, TEE\_Realloc, TEE\_Free (GP API as wrappers on top of POSIX API)

The memory allocation model of TA is represented on figure as below.

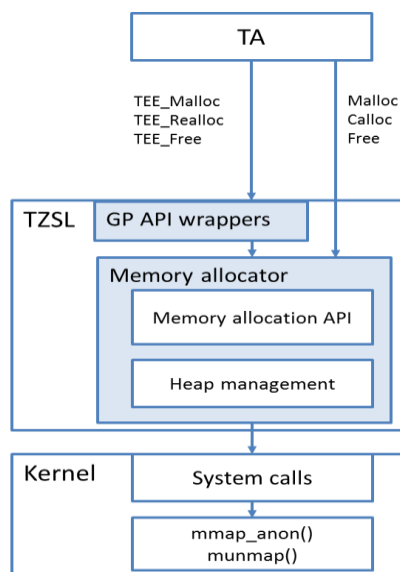


Figure 8 Memory Allocation

### **4.5.2 Inter-world shared memory**

Inter-world shared memory (iwshmem) is memory area accessible both from NWd and SWd. The memory itself comes from NWd userspace. It can be used by NWd to pass data to SWd and by SWd to return data in response to Nwd request.

iwshmem registration sequence:

1. NWd userspace allocates memory area
2. Finds physical pages corresponding to the iwshmem area
3. Allocates ID for newly created iwshmem
4. Calls SMC to register iwshmem info(ID and physical page address) in SWd
5. Secure Kernel saves info about iwshem for usage

## 5 Security Guideline

### 5.1 Assumptions on TA Development

TA developers are assumed to comply with the TA development guidelines set by the TEE provider.

- TA development should be done in a protected environment, following guidelines and by skilled persons being aware of the security issues.
- TA developers should not rely on protection of the TEE persistent data, TA data and keys, and TA code against full rollback
- TA developers are assumed to consider the following principles during the development of the Trusted Applications.
  - CA identifiers are generated and managed by the REE, outside the scope of the TEE. A TA must not assume that CA identifiers are genuine
  - TAs must not disclose any sensitive data to the REE through any CA (interaction with the CA may require authentication means)
  - Data written to memory that are not under the TA instance's exclusive control may have changed at next read
  - Reading twice from the same location in memory that is not under the TA instance's exclusive control can return different values.

For more information, see <https://www.globalplatform.org/specificationform.asp?fid=7831>

### 5.2 Persistent Object

TEE\_CreatePersistentObject in TEE\_Internal Core API spec does not specify format of initialData buffer, meaning that initialData can be zero terminated string, non-zero terminated string or any binary data. TEEGRIS TEE will check whether the malicious attackers are crossing across the virtual address area (VMA) by manipulating initialData buffer.

However, TEEGRIS does not always guarantee such behavior due to the nature of generic memory subsystem. TEEGRIS only partially guarantee it within the single virtual address area. If initialDataLen is closed to the legitimate length of initialData (e.g., the legitimate length + 1), TEEGRIS does not guarantee that initialDataLen is valid for the initialData buffer. This behavior is reasonable in terms of complier point of view, since declared buffer can reside in data section, with neighboring legitimate data. It is under the application developer's responsibility to carefully define and ensure that initialDataLen matched to the actual size of initialData buffer.

### 5.3 [outbuf] in Cryptography

Any DoFinal and Asymmetric function in GP API Cryptography, both buffers of destination (\*buffer) and length of destination (\*size) should be following [outbuf] annotation. TEEGRIS have two restrictions on [outbuf] annotation: (1) The buffer should be allocated by the calling application, and must be entirely writable by this application otherwise TEEGRIS TEE will panic the calling application, (2) If the output does not fit in the output buffer, then TEEGRIS TEE will update \*size with the required number of bytes (depending on cryptographic algorithm), and return TEE\_ERROR\_SHORT\_BUFFER. For example, TEE\_AE\_DecryptFinal has [outbuf] annotation on \*destLen. If the \*destLen is shorter than the expected size during accumulation, It return TEE\_ERROR\_SHORT\_BUFFER.

### 5.4 Hardware Cryptography

TEEGRIS provides the TEE\_AllocateOperation with custom extension of Hardware cryptography. This can enrich user experience with performance and security. Since the specification of Hardware cryptography depends on System-On-Chip (SOC) vendor, the application developers should consult with TEEGRIS TEE owner in terms of availability.

---

## Samsung TEEGRIS

In addition, TEEGRIS TEE doesn't recommend DES, and MD5 cryptography algorithm due to security reason (i.e., end of lifecycle). The application developers should not intent to use such algorithms.

### 5.5 Secure Object

TEEGRIS provides `SO_AccessControllInfoType` for wrapping secure object created by an application. Basically, `access_flags` defines what `ta_id` and `auth_id` can be permitted to unwrap the generated secure object.

It is under the application developers's responsibility to carefully designate that `access_flags` is well-matched to the intention. TEEGRIS also recommend that the application developer carefully consider to append `AUTH_ID_AC` as well, if they designate only `TA_ID_AC` for `access_flags`. It can prevent same `ta_id` from different `auth_id` from decrypting the specified secure object. The developer should not designate `NULL`, or invalid `access_flags` as it is weak protection.

TEEGRIS provides an API to help application developers to check such validity by `TEES_CheckSecureObjectCreator`. The developer should check the origin of secure object via `TEES_CheckSecureObjectCreator`.

### 5.6 Obfuscation of TA

Since Trusted Applications are not protected in confidentiality by default, developers must consider obfuscating the code of Trusted Applications if disclosing their internal functioning can be considered as security breach.

### 5.7 Stack Protector for TA code

TEEGRIS SDK enables stack protection by default and compiles TA code with "strong" (`-fstack-protector-strong`) protection version to defend most vulnerable functions against stack corruption attacks. Developers are able to override default toolchain option and protect every function using "`-fstack-protector-all`", however this option may lead to significant impact on code size and performance.

### 5.8 Parameter handling

Any parameters from NWd to the TA interface should be checked whether each of them has a designated type or not. As we provide `TEE_PARAM_TYPES` macro, so that TA developer can check whether an incoming `paramTypes` is matched to the type of all the desired parameters. You will need this check on both `TA_OpenSessionEntryPoint` and `TA_InvokeCommandEntryPoint`.

TA developers may want to check whether TA has the access right on buffer address at parameters. Though this is redundant in TEEGRIS architecture if `paramTypes` are checked, but we provide `TEES_IsREESharedMemory` API for those who may seek an equivalent API to other TEE platforms.

## 6 GlobalPlatform Certificate

This section presents how TEEGRIS provides the GlobalPlatform certified function.

### 6.1 Protection profile

Globalplatform is an industry association driven by over 100 member companies, as working on a standard of the secure management of card and devices once deployed in the field. Globalplatform has defined Protection Profile (PP) and published by ANSSI organization.

The PP defines the following security functional components

- FAU\_ARP.1 Security alarms
- FAU\_SAR.1 Audit review
- FAU\_STG.1 Audit event storage
- FCS\_COP.1 Cryptographic operation
- FCS\_RNG.1 Random numbers generation
- FIA\_ATD.1 User attribute definition
- FIA\_UID.2 User identification before any action
- FIA\_USB.1 User-subject binding
- FDP\_ACC.1 Subset access control
- FDP\_ACF.1 Security attribute based access control
- FDP\_IFC.2 Complete information flow control
- FDP\_IFT.1 Simple security attributes
- FDP\_ITT.1 Basic internal transfer protection
- FDP\_RIP.1 Subset residual information protection
- FDP\_ROL.1 Basic rollback
- FDP\_SDI.2 Stored data integrity monitoring and action
- FMT\_MSA.1 Management of security attributes
- FMT\_MSA.3 Static attribute initialization
- FMT\_SMR.1 Security roles
- FMT\_SMF.1 Management functions
- FPT\_FLS.1 Failure with preservation of secure state
- FPT\_ITT.1 Basic internal TSF data transfer protection
- FPT\_STM.1 Reliable time stamps
- FPT\_TEE.1 Testing of external entities
- FPT\_INI.1 TSF initialisation

### 6.2 Certified TEEGRIS

One of TEEGRIS version has been evaluated by Globalplatform accredited laboratory thoroughly.

- All TEE API (including TEE Client API, and TEE Internal Core API) security is also verified by Globalplatform accredited testing product (FIME)
- TEEGRIS has then been found resistant to attacks performed by an attacker possessing TEE-Low attack potential, which is at EAL2+ level

You can find our certified TEE at <https://www.globalplatform.org/teecertificationprod.asp>

Beyond this certification, Samsung continuously demonstrate TEEGRIS upcoming version to conform with upcoming Protection Profile as well.