SAMSUNG

Samsung TEEGRIS
Secure OS overview

Author	m.sec.teesdk@samsung.com	Date/Version	2018-09-18 TEEGRIS v3.0
Organization	Mobile Security Technology group, Samsung Electronics		

1 Introduction 3
1.1 Objective 3
1.2 Scope 3
1.3 Abbreviations and Acronyms 3
2 TEEGRIS architecture 5
2.1 Client Application 6
2.2 TEE Client library 6
2.3 TrustZone and TrustZone inter-world socket drivers 7
2.4 Secure kernel 7
2.5 Secure libraries 8
3 TEEGRIS feature 10
3.1 Loadable drivers 10
3.2 Over-The-Air 11
3.2.1 OTA for dummies (downloadable TA) 11
3.2.2 TA Installation 11
3.3 Trusted User Interface 12
3.3.1 Samsung TUI Technical Details 12
4 TEEGRIS API 16
4.1 GP TEE API 16
4.1.1 GP TEE Client API 16
4.1.2 GP TEE Internal API 16
4.1.3 GP TUI API 16
4.1.4 GP TEE Socket API 16
4.1.5 GP TEE Debug API 16
4.2 Specific API 17
4.2.1 Samsung internal API 17
4.2.2 Samsung Client API 17
4.2.3 Custom Handler API 17
4.2.4 SPI / I2C API 17
4.2.5 IRS API 17
4.2.6 Loadable Driver API 17
4.2.7 Thead support API 17
4.2.8 Math API 18
4.2.9 Auxiliary API 18
4.2.10 Hardware Crypto 18
5 TEEGRIS SDK 19
5.1 Prerequisite 19
5.2 Simple development 19
5.2.1 TEEGRIS SDK structure 19
5.2.2 Compilers list 20
5.2.3 Clang and Gcc 20
5.2.4 Hints and side-effects of Clang usage. 21
5.2.5 How to run the TEEGRIS SDK examples 21
5.3 Debugging techniques 22
5.3.1 Gettin g core dump on TA panic 22

1 Introduction

Samsung TEEGRIS is a system-wide security solution, which allows you to run applications in a trusted execution environment based on TrustZone. We present the TEEGRIS architecture for external developers to enable their trusted applications and services. Samsung Mobile uses the TEEGRIS for several commercial projects. You can utilize the TEEGRIS to run and deploy your applications.

TEEGRIS supports TrustZone technologies and guarantees the strengths of security (i.e., hardware cryptography, binary encryption, access control) and performance (i.e., multicore, multithreading). In addition, TEEGRIS provides a variant of GlobalPlatform APIs such as Trusted User Interface, Trusted Storage, and so on. TEEGRIS provides end-to-end service deployment environments so you can write, manage, deploy trusted applications on TEEGRIS.

1.1 Objective

This document provides overview of the TEEGRIS, which is intended to depict different aspects of the operation system and trusted application development.

1.2 Scope

The purpose of this document is a cursory examination of software components of the TEEGRIS, describe interfaces, and define data and control flow between components.

1.3 Abbreviations and Acronyms

Terminology / Abbreviations	Description
TEE	Trusted Execution Environment
TEE Client API	API to provide client interaction with Secure World using GlobalPlatform TEE Client API standard
TZ daemon	Rich OS system process (daemon) communicating with TZDev driver
TZDev driver	Linux kernel driver which provides access to TrustZone
Secure Kernel	OS Kernel running in Secure World
Secure Service Library API	API of set of TEEGRIS libraries for TEE OS
TrustZone (TZ)	ARM TrustZone is ARM HW security extension intended for TEE implementation
Client Application (CA)	An application running in Rich OS (ex/ Android, Tizen) and utilizing TEE Client API to access facilities provided by Trusted Applications inside the Trusted Execution Environment.
Trusted Application (TA)	An application running inside the Trusted Execution Environment that provides security related functionality to Client Applications outside of the TEE.
Execution Environment(EE)	An Execution Environment, as defined in OMTP ATE TR1 [1], is a set of hardware and software components providing facilities necessary to support running of applications.
Normal World (NWd)	An environment that is provided and governed by a Rich OS, potentially in conjunction with other supporting operating systems and hypervisors; it is outside of the TEE. This environment and applications running on it are considered un-trusted.

Secure World (SWd)	An execution environment: that runs alongside but isolated from a Normal World. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly.
Loadable driver (Driver TA, DTA)	An application, running inside the Trusted Execution Environment, that provides POSIX-like file operations for other TAs.
IRS	Integrity Reporting System
OTA	Over-the-Air
REE	Rich Execution Environment

2 TEEGRIS architecture

Mobile devices have powerful computing capability, various sensors and always connected. Many services and applications (ex. online banking, movie and pay) are working on mobile devices by utilizing such features. However, such feature could be exposed to many security issues.

To solve the problems and present TEE (Trusted Execution Environment), Samsung designed its own Secure OS, and named it as TEEGRIS. TEEGRIS is a Secure OS which utilizes the ARM TrustZone technology by provide TEE for Trusted Applications (TA). Main software components of TEEGRIS OS are: secure kernel, secure service libraries and rich OS communication modules.

Rich OS in normal world and TEEGRIS in secure world have to communicate with each other. The communication model is represented on the picture below. The Client application sends the request to the TZDev driver in the Rich OS kernel space. TZDev driver performs calls to pass control to the TEEGRIS secure kernel. The TEEGRIS secure kernel then switches the execution to the trusted application for the client's request execution. After the request execution is completed, the workflow moves back to the TEEGRIS secure kernel, then to the Normal World (TZDev driver) and finally to the client application.

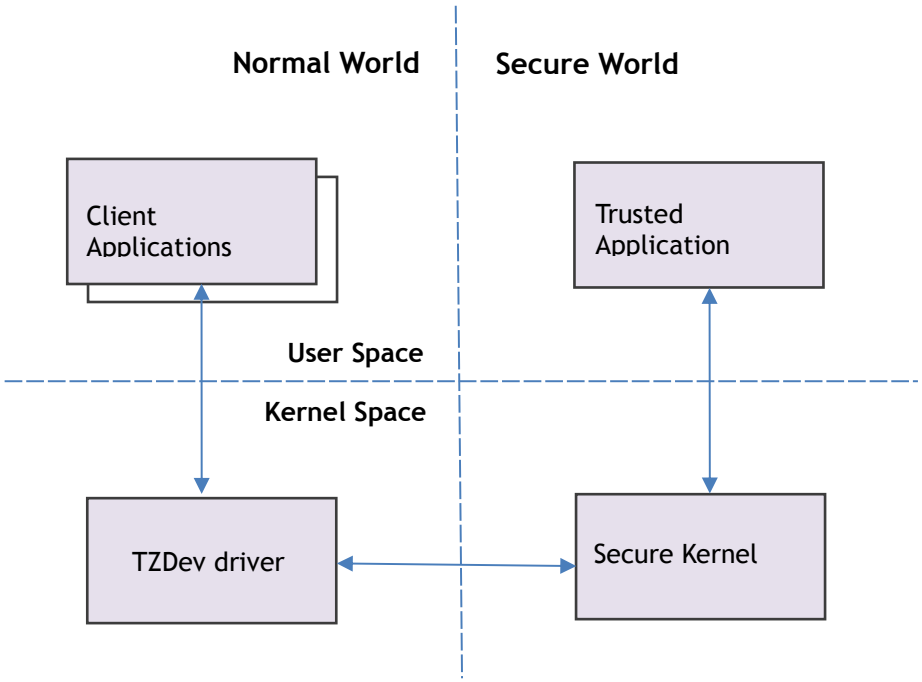

Figure 1 TEEGRIS Architecture

TEEGRIS offers a number of technical and commercial benefits to developers and end-users.

These include:

- A new level of safe environment for secure data and code. This enables a complete approach to security. For example, hiding peripherals and memory from the non-secure world provides HW level of secure data protection like DRM content, crypto keys and personal data from malicious attack or unauthorized access.
- Because the solution consists of software and hardware elements, it provides flexibility to allow customization and upgrades to the secure SW after device is released.

The diagram below represents scheme of communication and interaction between software components.

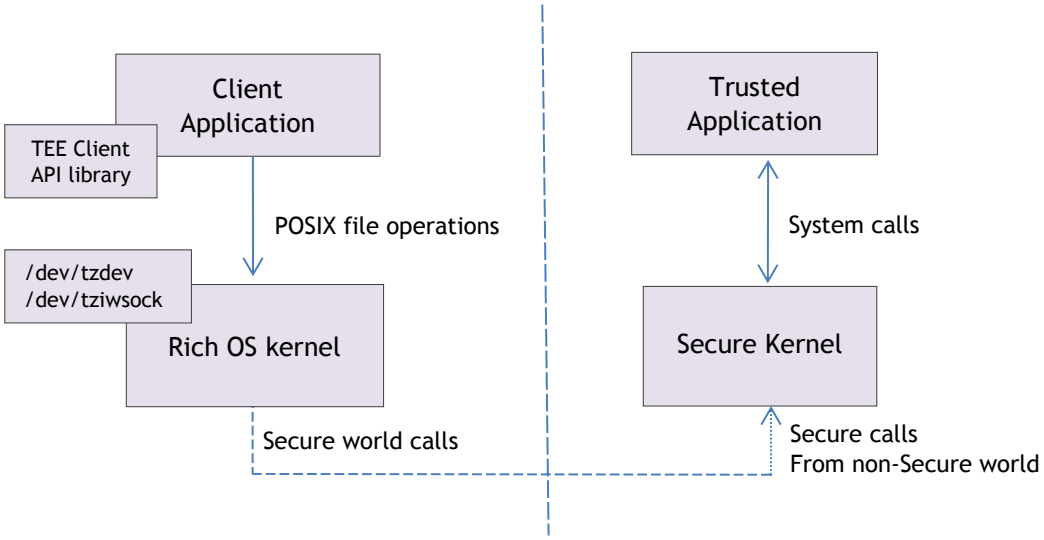

Figure 2 Communication and Interaction Schema

2.1 Client Application

An application in the Normal World which uses the TEE secure services (i.e. accesses to Trusted Application). Its design and purpose are defined by the application developer and is out of scope of this document.

Functions

Client application has to perform the following functions:

- Communicate with TEE Secure Service
- Conform to TEE Client API specification requirements

Processing

Defined by application developer, the Secure Client workflow should not violate the TEE Client API specificationzrequirements.

2.2 TEE Client library

A Normal World library, which implements TEE Client API functions.

Functions/ Processing

TEE Client API has to perform the following functions:

- TEE context initialization/finalization
- TEE shared memory areas registration/release
- TEE session starting/closing
- Command sending to Secure World
- TEE Client API library should implement the following API functions as described in TEE Client API specification:

    - TEEC_InitializeContext
    - TEEC_FinalizeContext
    - TEEC_RegisterSharedMemory
    - TEEC_AllocateSharedMemory
    - TEEC_ReleaseSharedMemory
    - TEEC_OpenSession

- TEEC_CloseSession
- TEEC_InvokeCommand

More information, see the GlobalPlatform Specifications
http://www.globalplatform.org/specificationsdevice.asp

TEE Client API functions' calls communicate with TrustZone driver (TZDev)

2.3 TrustZone and TrustZone inter-world socket drivers

TrustZone driver (/dev/tzdev) and TrustZone inter-world sockets driver (/dev/tziwsock) provide an interface to the ARM TrustZone functionality as this is based on the POSIX file operations.

- Shared memory areas passing to secure world
- CPU frequency booster
- Crypto clock management and others

The TrustZone inter-world sockets driver provides ioctl interface that simulate POSIX sockets API. This device is responsible for communication between NWd and SWd processes.

※ Your application should have the appropriate rights to access those two device drivers.

2.4 Secure kernel

OS kernel implementation with functionality, intended to perform system-level operations (memory management, scheduling, device drivers) in secure world.

Functions

Secure Kernel has to perform the following functions:

- Implement mechanisms for memory and devices protection from Non-Secure World access
- Memory management
- Tasks scheduling
- Inter-Process communications (IPC)
- Implement device drivers functionality and namespace management
- Handling system calls from Secure userspace
- HW resources management (interrupts, timers, etc)

Input

After loading, the Secure Kernel is invoked by a call passed from the Normal World to the Secure World from the TrustZone inter-world sockets driver. Secure kernel implements handlers for processing calls from NWd (NWd handlers). Each NWd handler could accept several arguments passed to it, via registers, and operate with shared memory areas by reading and writing data from/to them.

Processing

The Secure Kernel behavior could differ depending on NWd call handler purpose and implementation. Depending on NWd call, the Secure Kernel could perform one or more of the following actions:

- Register shared memory area
- Release shared memory area
- Start Trusted application

- Route command to Trusted application
- Schedule Trusted Application tasks
- Stop Trusted application
- Handle requests to device drivers

Output

On returning to the NWd Secure Kernel passes return code for NWd call it was invoked by. Beside that shared memory areas could contain modified data, depending on NWd call implementation. Also, NWd call handlers may change behavior of Trusted Applications by starting, blocking/unblocking and stopping them.

2.5 Secure libraries

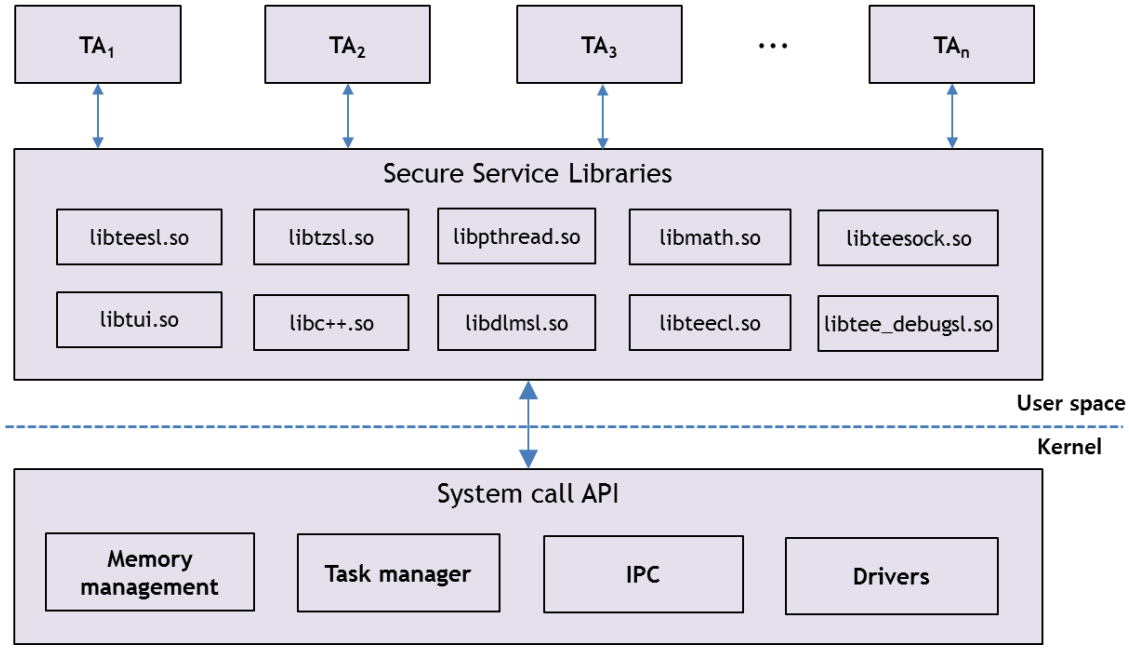

Figure 3 Secure Service Libraries View

The System libraries add an abstraction layer from the Secure Kernel and allow users to work with high-level primitives instead of calling system calls to the Secure Kernel directly.

By default, the user TA is linked with libteesl.so and libtzsl.so. TEEGRIS SDK contains stubs for system libraries, so the user can use library stubs for linking. If needed, the user TA can be linked with other libraries (libmath.so, libteesock.so), in this case, the required library should be added to the linker call with -l flag (-lmath, -ltee_sockets).

The set of libraries are split by their functionalities: POSIX-like generic API, TEE API, mathematic API, TEE Socket API. Due to the following different standards, system libraries have different code style convention and error handling mechanisms. Be careful with different error handling mechanisms.

Libraries:

- libtzsl.so

Provides POSIX-like API for generic functions like open, ioctl, printf... For error handling library uses errno value, which is defined in errno.h header file. errno value is thread-safe, and can be successfully used from different application threads. Each thread will have its own copy of errno. The most of functions, defined in this library can have documentation in Linux man pages, so it can

be found by typing man <function name> in the Linux terminal. Detailed documentation of this library also can be found in the "Samsung Internal API Guide.doc" document.

- libteesl.so

Provides the implementation of the TEE Internal API functions, userspace driver API and Samsung extension of the TEE Internal API (functions starts from TEES_). For error handling, functions return TEE_Result value which contains the reason of error. All possible error codes can be found in the file tee_internal_api.h which is present in the TEEGRIS SDK. Functions, defined in this module are NOT thread-safe, and cannot be called from different threads. Detailed documentation of the TEE Internal API function set can be found at the GlobalPlatform portal:
http://globalplatform.org/specificationsdevice.asp
Detailed documentation of the Samsung extension of the TEE Internal API and userspace driver API can be found in the "Samsung Internal API Guide.doc" document.

- libpthread.so

Provides implementation of thread management functions. The set of implemented functions are compatible with standard POSIX functions, so it is possible to find their documentation in the Linux man pages with: man <function name>. For error handling each function returns an error code. Available error codes can be found in the errno.h file of the TEEGRIS SDK. Functions, defined in this module does not set errno variable. Detailed documentation of this library also can be found in the "Samsung Internal API Guide.doc" document.

- libteesock.so

Provides implementation of the TEE Socket API functions which is used for socket-like communication between the TA and normal world applications. For error handling, functions return the TEE_Result value. All possible return codes can be found in the tee_internal_api.h file from the TEEGRIS SDK. Detailed documentation of the TEE Socket API function set can be found at the GlobalPlatform portal: <
http://globalplatform.org/specificationsdevice.asp

3 TEEGRIS feature

3.1 Loadable drivers

Loadable drivers in TEEGRIS are implemented in a concept of user-space driver. In many aspects, the user-space driver is very similar to ordinary user-space process, i.e. Trusted Application (TA):

- It is built separately from the Secure Kernel code, using a set of libraries included in the secure toolchain
- It can be loaded and unloaded (e.g. terminated) at run-time
- It runs in a separate address space, so faults and exceptions occurred in the user-space driver cannot damage the secure kernel and other TAs, and can be easily managed by the Secure Kernel

Despite the user-space driver being similar to the ordinary user-space process, it has to have the following additional properties:

- Higher level of privileges (physical memory and registers mapping)
- Ability to register IRQ handler(s) or subscribe to IRQ notification
- Ability to register device name in namespace subsystem

The Secure Kernel provides the following functionality to allow implementation of the user-space driver model:

- API to register device name and mask of file operations to namespace subsystem at run-time from the user-space
- API to register IRQ handler(s) or subscribe to IRQ notification at run-time from the user-space
- Routing of file operations' requests from one TA to another and back to requesting TA
- Handle attempts to use file descriptors to the failed user-space driver

As a built-in (kernel-mode) driver of the Secure Kernel, the user-space driver provides the following functionality:

- Device name in namespace subsystem, which can be accessed by TAs using POSIX file operations (open, close, read, write, mmap, ioctl or their subsets)
- Maintain POSIX-like file operations
- Able to manage requests from several TAs, accessing it simultaneously

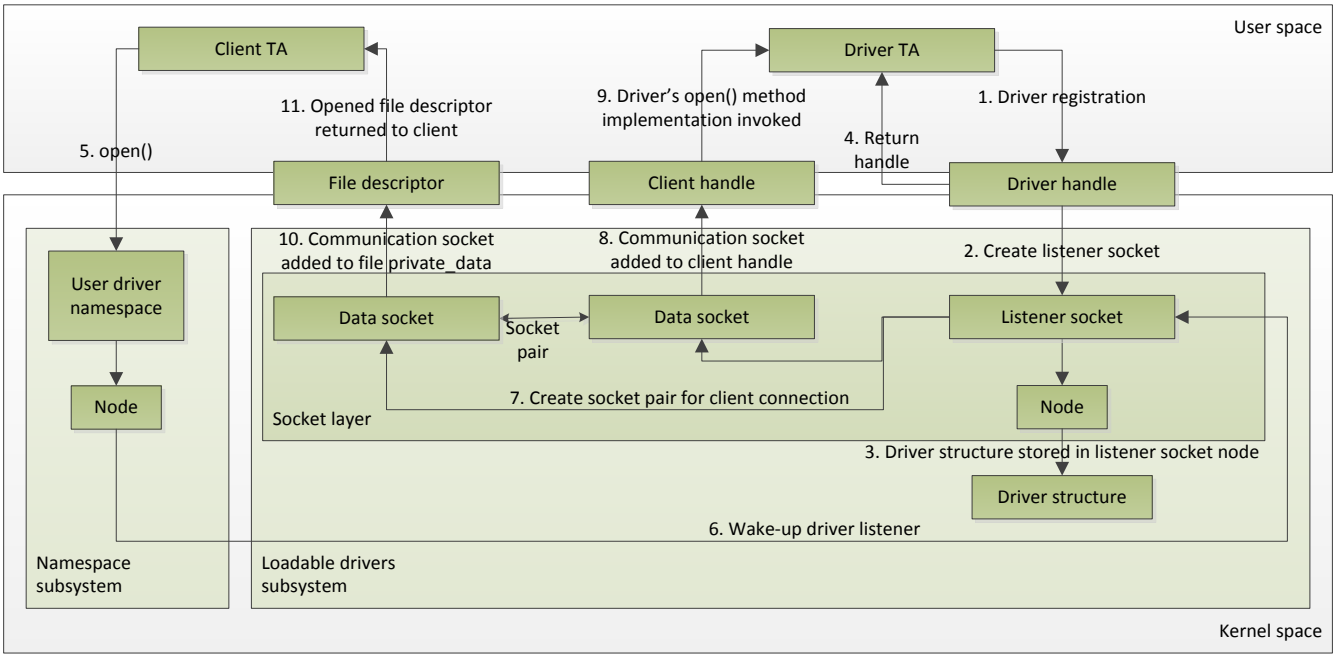

Figure 4 User-Space Driver Architecture

3.2 Over-The-Air

An operating system is void and useless without applications running atop of it and secure operating systems like TEEGRIS are not an exception. Yet unlike normal world OS, TEEGRIS has much more strict security requirements that make it difficult to combine possibility to use any 3rd-party Trusted Applications (TA) from one side and still maintain a high level of security from another. No malicious TAs should be allowed on the system which means that every Trusted Application should be first checked and authorized to run on a device.
To allow secure installation and management of 3rd-party TAs, TEEGRIS Over-the-Air TA installation subsystem was designed. It supports downloading of TAs from either Google Play service or dedicated OTA server run by Samsung. Policy that regulates rules applied to 3rd-party TAs can be dynamically updated to allow quick reaction to changing business needs.

3.2.1 OTA for dummies (downloadable TA)

Regardless of OTA functionality's readiness, TEEGRIS can support downloadable TA in a simple way. Custom API TEECS_OpenSession can be designated with buffer reading from file descriptor. TA developer can put their code into either REE application package or even the dynamic buffer itself.

3.2.2 TA Installation

3rd-party TAs can be downloaded from Google Play as part of apk, or later acquired from OTA server by OTA Agent daemon, an Android service responsible for OTA operation.

Two ways are available for TA installation over-the-air: via Google Play service and via OTA server. In order to be able to use Google Play service for TA installation the TA should be located in raw resources of the apk (/res/raw folder of apk). After the application containing TA is downloaded and installed, a broadcast message is sent to notify the system about the new application. This broadcast message is received by the OTA Java service that runs with system privileges. The OTA Java service copies the TA from the APK resources into the system folder where it can be found by TEEGRIS. After that, the TA will be started when requested by the client application.

Another way to install the TA into the system is downloading it from the OTA server. This way, it will work both for Java and native clients. When a client requests the start of a TA not present in the system, the OTA Agent daemon is notified about this error. It connects to the OTA server and requests the download of the missing TA. If the TA is present in the OTA server's database and this TA is allowed to run on the given device the TA is downloaded and installed to the device. After that request to start, the TA will automatically be repeated and the TA will be launched assuming that the corresponding records in ACSD are present.

※ Application developer should contact theo TEEGRIS administrator regarding the feasibility of OTA server at target platform.

3.3 Trusted User Interface

TEE based Samsung Trusted User Interface (TUI) solution provides hardware-isolated trusted environment suitable for secure input and secure display of sensitive information. The access to the TUI service is available only from TEE applications (TAs). So the plain sensitive data can be handled only in the TEE and is never available outside the TEE.

The basic goal of the TUI is to secure interaction with user. To achieve that goal, TUI provides the following basic security aspects:

- Secure Display
  - Frame buffer protection and cleanup
  - Display device protection
  - TUI resources (button images, keypad images, etc.) protection
- Secure Input
  - Touch Device Protection
- Secure Indicator
  - Indication of TUI session by dedicated hardware device (ex. LED controlled by TEE only), or
  - Indication by user personal information or dedicated image if hardware device is not available.

3.3.1 Samsung TUI Technical Details

TUI components in TEEGRIS are shown in Figure 5. TUI CA uses the GP client API to open context and sessions for TUI TA, display driver TA, touch driver TA. TUI TA is dynamically linked to TEEGRIS TUI library, where 2 kinds of TUI APIs are supported. GP TUI APIs conforming to GP TUI 1.0 specification are defined in tui.h for GP TUI session or Samsung internal TUI APIs are provided for low level TUI session in TUI lib.

First of all, if you prefer using GP TUI APIs, you don't have to worry about the predefined or hardcoded GP style of the TUI screen, which may be different from your UX requirement. TEEGRIS provides a GUI tool in Figure 6 to configure the TUI screen as you like and sign it into resource files. You may set the hierarchy, action for input event as well as dimension, position for each UI component with it. Then the GUI engine, based on object oriented programming of TEEGRIS TUI lib, parses that resource file and works as specified. Any engine code change is not needed at all for different layouts. For example, you may easily meet a complex UX scenario like hierarchical keyboard for small form factor devices shown in Figure 7. Though the main usage of TUI is PIN entry, our TUI solution may extend it into various SWD GUI applications like below.

- Banking application : authentication data, payment
- Browser : passwords
- Messaging : SMS, MMS, email, instant messages

- Organizer : tasks, calendar
- Health application : health data/history

Either landscape or portrait layout is selectable for each session.

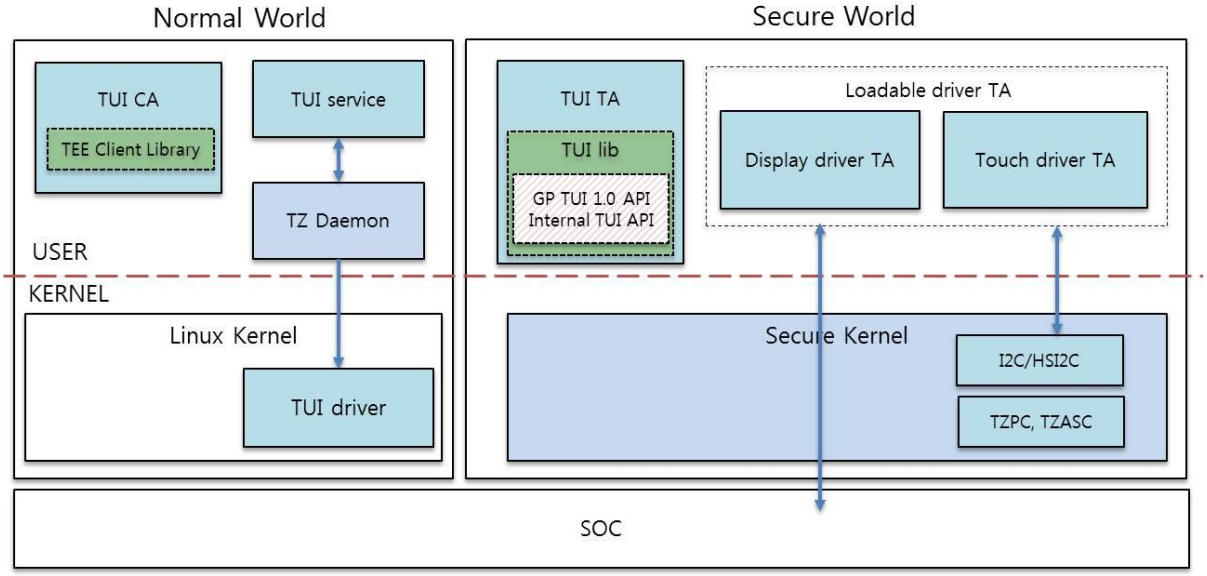

Figure 5 TUI Components

Secondly, if you have your own common implementation for TUI across various secure OS; the internal TUI APIs in tees_tui.h may help in that low level TUI session case. This would manage needed images for the TUI screen in your own way, e.g. including them inside TA, and replace basic low level TUI APIs in 1-to-1 API mapping. Internal TUI APIs supports portrait mode with text rotation.

The display driver TA and touch driver TA provides access to the specific hardware, i.e., display and touchscreen from TA. In the current design, TUI secure drivers are implemented as loadable device drivers. I2C/HSI2C driver, TZPC driver, TZASC driver reside in secure kernel and driver TA may map SFR regions for other HWs such as display controller, GPIO and control them in user space. More efficient implementation is available by using TEEGRIS interrupt API, for example, getting touch event.

The display driver TA and touch driver TA will be selectively available upon the requirement of TEE feature on specific platform. Once TEE user has these two drivers in the target platform, TEE user can fully see integrated TUI functionality and develop the application to utilize TUI including secure display, touch and indicator.

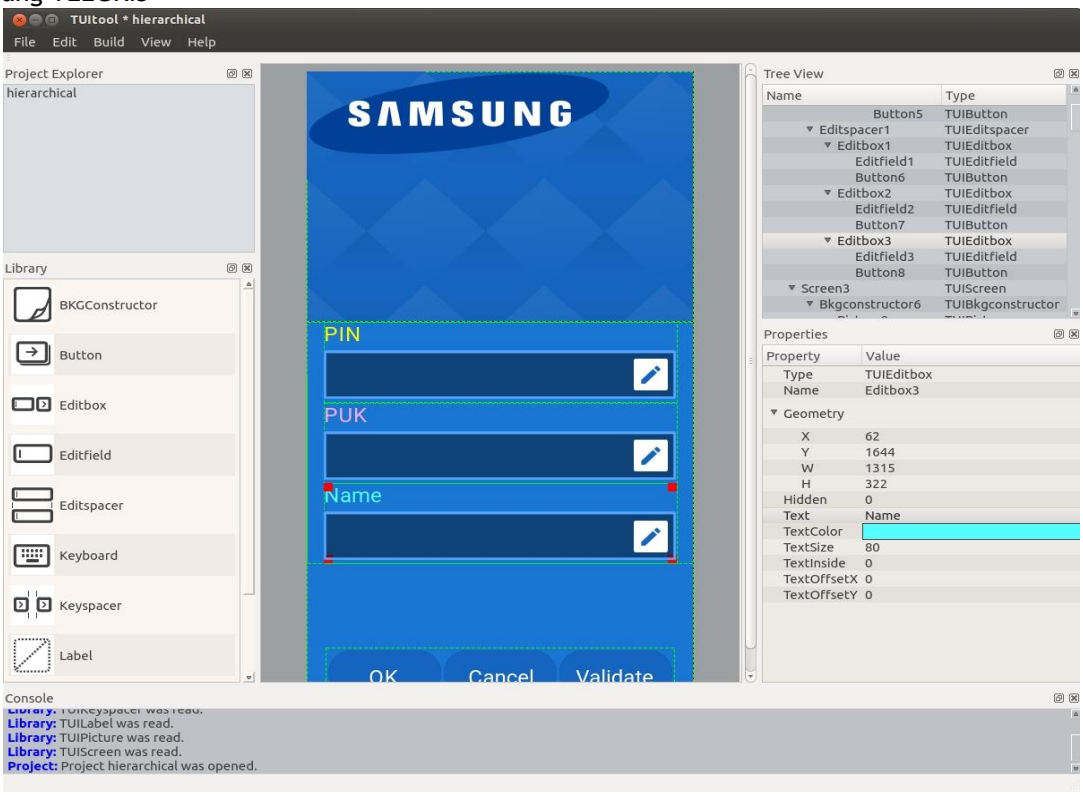

Figure 5 TUI Tool

Figure 6 Hierarchal Keyboard Example

The TUI driver in the Linux kernel prevents the NWD from accessing the TUI resources like controller or clock during TUI session, which may lead to bus error or crash. It deactivates the Linux display, touch drivers before TUI session starts and SWD driver initializes. It allocates frame buffer, work buffer and passes that information to SWD.

TUI service interrupts the TUI session in case of important events, like incoming call, SMS, platform abort or power management operations.

TEEGRIS TUI solution is layered software architecture for simplicity of porting to any platform. Linux TUI driver, display driver TA, touch driver TA, some of the secure kernel drivers need to be ported to enable TUI in devices. We may provide the reference code for Linux TUI driver, display driver TA, touch driver TA and TUI service. So does the TUI tool for custom TUI screen. Secure kernel changes will be included in the device firmware.

From a security consideration, the hardware modules (used for input or output information) are locked for TEE use only. In this case, the touchscreen and display becomes unavailable for any REE

software. The memory used for frame buffer, TUI resources or input data is also hardware protected and unavailable for REE access. So there is not any possibility for malicious software in REE to get access to the data showed on the display or tamper information inputted by the user. To prevent modification or substitution of TUI resources stored in REE file system they are protected by cryptographic algorithm. A broken TUI resource cannot be used in the TUI session.

Note that the TUI excludes the security of the applications themselves. Ultimate responsibility for data protection relies on the application developer and requires that applications be designed (or redesigned) to utilize the TEE interface and to do so properly that sensitive data is not exposed outside the TEE.

To sum up, the TUI solution provides secure way to get and display user's sensitive information on mobile devices. The information cannot be tampered with having any permissions and possibilities in REE. Samsung TUI guaranties safe interaction with user but can't prevent potential leaks of information in client applications itself due to possible issues in their design or implementation.

4 TEEGRIS API

The goal of this section is to describe the Secure OS and Service Library API which is used from TEEGRIS. The intended audience is Trusted Application (TA) developers with permission to access Samsung Internal API.

4.1 GP TEE API

The Global Platform API is set of API applied at TEEGRIS for development of TA running inside a TEE. It is based on GlobalPlatform specification.

4.1.1 GP TEE Client API

https://www.globalplatform.org/specificationform.asp?fid=7339
This specification defines a communications API for connecting Client Applications running in a rich operating environment with security related Trusted Applications running inside a Trusted Execution Environment.

4.1.2 GP TEE Internal API

https://www.globalplatform.org/specificationform.asp?fid=7857
This specification defines a set of C APIs for the development of Trusted Applications (TAs) running inside a Trusted Execution Environment. The APIs defined in this document target the C language and provide the following set of functionalities to TA developers:

- Basic OS-like functionalities, such as memory management, timer, and access to configuration properties
- Communication means with CAs running in the Rich Execution Environment
- Trusted Storage facilities
- Cryptographic facilities
- Time management facilities

4.1.3 GP TUI API

https://www.globalplatform.org/specificationform.asp?fid=7779
Certain applications need to expose sensitive information to the user for validation or to get sensitive information from the user. This document defines and specifies a Trusted User Interface (TUI) API for Trusted Application developers.

4.1.4 GP TEE Socket API

https://www.globalplatform.org/specificationform.asp?fid=7834
The GlobalPlatform TEE Sockets API Specification, specifies:

- The generic C interface used by a Trusted Application (TA) to establish and utilize network communications to a remote server using a socket style approach
- Generic error behavior of the functions

This general specification does not specify any particular protocol nor the data structures used to utilize a particular protocol. Each specific protocol, such as TCP and UDP, is described in detail in a separate Annex to this general specification.

4.1.5 GP TEE Debug API

https://www.globalplatform.org/specificationform.asp?fid=7856
This document specifies the GlobalPlatform Trusted Execution Environment (TEE) Debug interfaces and protocols. This version specifies:

- The syntax and semantics of the TA Post Mortem Reporting (PMR) client interface, i.e. local commands, and data structures transported through the TEE Client API to report the Panic state of a TA to a client application.

- The syntax and semantics of the Debug Log Message (DLM) client interface, i.e. local commands, and data structures transported through the TEE Client API to transfer debug “printf()” type messages to a display or logging tool.
- The syntax and semantics of the DLM internal interface, i.e. local commands, and data structures available to a TEE TA to make use of the DLM service.

4.2 Specific API

4.2.1 Samsung internal API

Samsung Internal API is Samsung's own API to support any privileged features to internal TA developers such as driver API, interrupt API, custom SMC handler, and so on. Only TA developers with permission to the API can use it, and 3rd-party TA developers should be prevented from accessing the API for security reasons.

4.2.2 Samsung Client API

Samsung Client API is Samsung's own API to support any privileged features to internal CA developers. Samsung Client API extends TEE Client API and it provides functionality for setting crypto clocks, modified version of open session function, TEE TA Debug functionality and getting description of errors code.

4.2.3 Custom Handler API

ARM Trust Zone technology provides possibility of Trusted Execution Environment (TEE) creation. As TEE is intended for execution of security critical applications and handling of security sensitive data. It is naturally that TEE Operational Systems (OSes) are developed with the aim of maximum stability, reliability and safety.
These are based on robust kernel with minimum needed functionality and Trusted (user) Applications (TAs) that don’t have any possibilities (privileges) to effect on TEE security.
With such design, it is impossible to extend Secure OS kernel (privileged) functionality in case of need as it may affect the basic features of Secure OS and leads to a lot of additional verification costs. At the same time unprivileged level only is not enough for some applications especially considering that their functionality is not needed for Secure OS work. To overcome this collision TEEGRIS Secure OS provides Custom SMC Handler feature which allows running some code in privileged environment without much effect on Secure OS.

4.2.4 SPI / I2C API

TEEGRIS provides several APIs for access and control to th SPI controller in order to be able to communicate with peripherals.

4.2.5 IRS API

The main task of IRS is control of Software One Time Program flag setting according to different system agents' requests and providing flag state information when it is needed. TEEGRIS IRS APIs provide secure place accessible from the early stages of OS boot to store integrity check results and to save/get results to/from this store in a secure way.

4.2.6 Loadable Driver API

You can resist device and use it at run-time from user-space.

4.2.7 Thead support API

TEEGRIS support POSIX thread API for SMP environment. So you can make multithreaded programming by thread create, join, lock and so on.

4.2.8 Math API

TEEGRIS provides Math API to make calculations more convenient. You can calculate big data or trigonometric function by using this APIs.

4.2.9 Auxiliary API

Auxiliary API is a set of API to provide POSIX-like interface to TA developers. The feature coverage is limited to the minimum set of functions rather than fully supporting POSIX standard in Secure World. Currently, only standard IO functions and string operations are supported by the Auxiliary API.

4.2.10 Hardware Crypto

TEEGRIS supoort not only S/W crypto, but also H/W crypto. It can give you more faster and more safe crypto operation, when you use H/W crypto APIs.

5 TEEGRIS SDK

5.1 Prerequisite

You must fill out the 'Security Pledge' for TA development and TA signing.

5.2 Simple development

TEEGRIS SDK is represented as a handy tool for building Trusted applications. It consists of documentation with examples, platform deepened libraries and headers, compilers, signing tools and etc.

5.2.1 TEEGRIS SDK structure

There is a structure of the universal toolchain:

```
teegris-sdk
├─── docs
│    ├─── examples
│    ├─── html
│    ├─── man
│    ├─── Samsung_client_api.pdf
│    ├─── Samsung_internal_api.pdf
│    ├─── Samsung_TEEGRIS_Overview.docx
│    ├─── Samsung_TEEGRIS_Security_Architecture.docx
│    ├─── teescl-html
│    └─── teescl-man
├─── platforms
│    ├─── BF-2.0 -> TEEGRIS-2.0
│    ├─── TEEGRIS-2.0
│    └─── TEEGRIS-3.0
├─── platform.version
├─── sdk.dep
├─── toolchains
│    ├─── aarch64-secureos-gnueabi-clang_4_0_1-linux-x86
│    ├─── aarch64-secureos-gnueabi-gcc_5_2-linux-x86
│    ├─── aarch64-secureos-gnueabi-gcc_6_3-linux-x86
│    ├─── aarch64-secureos-gnueabi-gcc_6_3-windows-win32
│    ├─── arm-secureos-gnueabi-clang_4_0_1-linux-x86
│    ├─── arm-secureos-gnueabi-gcc_5_2-linux-x86
│    ├─── arm-secureos-gnueabi-gcc_6_3-linux-x86
│    └─── arm-secureos-gnueabi-gcc_6_3-windows-win32
├─── toolchain_version.txt
└─── tools
     ├─── add_manpath
     ├─── msys
     ├─── OpenSSL-Win32
     ├─── SecOS_Dump_linux_x64
     ├─── SecOS_Dump_win32
     └─── teegris_authority_scripts
```

Here is a more detailed structure of the platforms directory:

```
platforms
└──── teegris-3.0
    ├──── nwd
    │   └──── android
    │       ├──── arch-arm
    │       ├──── arch-arm64
    │       └──── includes
    └──── swd
        ├──── arch-arm
        │   └──── usr
        └──── arch-arm64
            └──── usr
```

Supported architectures are ARM for 32 and 64 bit and Android OS.
TEEGRIS SDK is distributed in archive TEEGRIS_SDK.date.tar.gz
For testing TEEGRIS SDK you should unpack the archive, build and run any example from docs/examples directory.

5.2.2 Compilers list

Which of the compilers to use?
There are list of compiler in SDK

```
...
├──── toolchains
│   ├──── aarch64-secureos-gnueabi-clang_4_0_1-linux-x86
│   ├──── aarch64-secureos-gnueabi-gcc_5_2-linux-x86
│   ├──── aarch64-secureos-gnueabi-gcc_6_3-linux-x86
│   ├──── aarch64-secureos-gnueabi-gcc_6_3-windows-win32
│   ├──── arm-secureos-gnueabi-clang_4_0_1-linux-x86
│   ├──── arm-secureos-gnueabi-gcc_5_2-linux-x86
│   ├──── arm-secureos-gnueabi-gcc_6_3-linux-x86
│   └──── arm-secureos-gnueabi-gcc_6_3-windows-win32
...
```

Status of each toolchain

gcc 5.2	deprecated and not recommended to use
gcc 6.3	recommended to support existing TA code
clang 4.0.1	preferable for development of new TA

5.2.3 Clang and Gcc

Pro's of clang vs GCC (text from https://clang.llvm.org/comparison.html):

- Clang is designed as an API from its inception, allowing it to be reused by source analysis tools, refactoring, IDEs (etc) as well as for code generation. GCC is built as a monolithic static compiler, which makes it extremely difficult to use as an API and integrate into other tools. Further, its historic design and current policy makes it difficult to decouple the front-end from the rest of the compiler.
- Clang does not implicitly simplify code as it parses it like GCC does. Doing so causes many problems for source analysis tools: as one simple example, if you write "x-x" in your source code, the GCC AST will contain "0", with no mention of 'x'. This is extremely bad for a refactoring tool that wants to rename 'x'.

- Clang is much faster and uses far less memory than GCC.
- Clang has been designed from the start to provide extremely clear and concise diagnostics (error and warning messages), and includes support for expressive diagnostics. Modern versions of GCC have made significant advances in this area, incorporating various Clang features such as preserving typedefs in diagnostics and showing macro expansions, but GCC is still catching up.
- GCC is licensed under the GPL license. clang uses a BSD license, which allows it to be embedded in software that is not GPL-licensed.
- Clang inherits a number of features from its use of LLVM as a backend, including support for a bytecode representation for intermediate code, pluggable optimizers, link-time optimization support, Just-In-Time compilation, ability to link in multiple code generators, etc.
- Clang's support for C++ is more compliant than GCC's in many ways.
- Clang supports many language extensions, some of which are not implemented by GCC. For instance, Clang provides attributes for checking thread safety and extended vector types.

5.2.4 Hints and side-effects of Clang usage.

On the one hand Clang has been designed to provide concise diagnostics and includes support for expressive diagnostics.
On the other hand, it can be course of new errors and warnings while changing compiler Gcc to Clang for existing TA source code. TA developer should be ready to fix additional bugs.
That is why we recommend TA developers to use Gcc for existing code and Clang for new development.

5.2.5 How to run the TEEGRIS SDK examples

Unpack the TEEGRIS SDK archive.

```
$ tar –xf TEEGRIS-SDK-3.0.tar.gz
```

Move to the TEEGRIS directory and set SDK/NDK path in config.mk

```
############################
##### user set space #####
############################
## 1. Set SDK / NDK PATH
CONFIG_SW_TOOLCHAIN_ROOT = /home/xxx/…/teegris_sdk
CONFIG_NDK_ROOT = /home/xxx/…/android-ndk-r14b
```

Move the example directory and run the makefile.

```
$ cd teegris_sdk/docs/examples/hello_world
$ make –f Makefile.Single
Building TA: 10000000-0000-0000-0000-000000000000
Building CA: hello_client
Build Done.
```

Output binaries is in “dist” directory.
If you can use adb and connect your device, you can upload binaries using script file.

```
$ sh push_android.sh
```

If you want to build all of example, move to “teegris_sdk/docs/examples” directory and run the makefile.

```
$ cd teegris_sdk/docs/examples
$ make –f examples.mk
```

To launch apps

```
$ adb shell /system/tee/hello_client
SWd responded by : Hello from SWd
```

In case of success client application returns phrase from Secure World. If something goes wrong, use dmesg or logcat for more detail.

```
$ adb shell dmesg | grep ‘SW’
```

5.3 Debugging techniques

5.3.1 Getting core dump on TA panic

TEEGRIS allows developers to get the core dump file. This dump can be analyzed with tools like **readelf**, **gdb**.
Make sure you have tools **arm-secureos-gnueabi-readelf** and **arm-linux-androideabi-gdb** included in PATH environment variable.

Tools are placed in next directory (for 32 bit):

```
<bf_sdk>/toolchains/arm-secureos-gnueabi-5.2-linux_x86/bin/arm-secureos-gnueabi-readelf
<android ndk root>/toolchains/arm-linux-androideabi-4.9/prebuilt/linux-x86_64/bin/arm-linux-androideabi-gdb
```

Tools are placed in next directory (for 64 bit):

```
<bf_sdk>/toolchains/aarch64-secureos-gnueabi-5.2-linux_x86/bin/aarch64-secureos-gnueabi-readelf
<android ndk root>/toolchains/aarch64-linux-android-4.9/prebuilt/linux-x86/bin/aarch64-linux-android-gdb
```

The other steps are the same for both architectures.
To get the core dump do next steps:

1 - on device, launch **tee_transport_read**

```
$ tee_box tee_transport_read
```

In case of success **tee_transport_read** prints

```
Ready to work
```

2 - on device, launch your TEE client application in another terminal. For example:

```
$ your_client_app
```

3- If TA panics tee_transport_read tool will create th core dump on the device in /data/tee/core_dump (Android) or /var/tee_core_dump (Tizen) directory and show name and path of created file.

For example:
data/tee/core_dump/core.1089.9367418541666
The structure of name is "core.<TA_pid>.<some_random_number>".

4 - get core dump file to host computer with **pull** command.

For Android

```
(host) $ adb pull /data/tee/core_dump/<your_core_dump> <host_folder>
```

For Tizen

```
(host) $ sdb pull /data/tee/core_dump/<your_core_dump> <host_folder>
```

Dump file can be opened with readelf tool:

```
arm-secureos-gnueabi-readelf -a <your core file>
```

or with gdb:

```
arm-linux-androideabi-gdb <path to non-signed TA> <your core file>
```

More information related to working with core dump
http://yusufonlinux.blogspot.ru/2010/11/debugging-core-using-gdb.html