Emulating Trust Zone in Android Emulator with Secure Channeling

Arun Muthu[1], Rahim Rahmani[2] and Dinakaran Rajaram[3]

[1] Department of Information and Communication Technology, Kungl Tekniska Högskolan (KTH), Stockholm, Sweden

[2] Associate Professor, Department of Computer & Systems Sciences (DSV), University of Stockholm, Stockholm, Sweden

[3] Department of Information and Communication Technology, Kungl Tekniska Högskolan (KTH), Stockholm, Sweden

Abstract

There is a raise in penetration of smart phone while using enterprise application, as most of them are downloaded from the public market, resulting in challenge for security framework, causing a threat to lose sensitive user data. To prevent this ARM introduces the virtualization technique in hardware level, which prevents processing of trusted application that is completely isolated from general processing. To improvise this, we need to understand ARM Architecture; however it is still black box for users and developers. In this article, we take a deep look at the hardware architecture of the ARM trust zone to study and analyze its implementation and also to create its replica in emulator. Moreover we describe feasibility of various designs, implementation of trust zone feature in android emulator; with sample trusted application called secure channeling and concludes with annotation of suitable design on future enhancement. The security domain for secure processing and utility in emulator is to benefit the user and developer community.

Keywords: *Trust zone, Emulator, Android, Virtualization, Security and secure channeling.*

1. Introduction

Technology seeking is expanding widely in all corners of the world and Smartphone is one among them. Now a days, we see plenty of smart phone users, progressively increasing in recent years and is expected to be more, since using smart phone is easy to access the application download, compatible and portable compared with laptop and notebook. However inability design and improper handling of the security critical functionalities of the Smartphone shows that no technology is resistant toward the security leak or attack till now. First android smart phone was introduced in the year 2008. Soon after the release, we found lot of security leaks and vulnerability in security architecture of the android OS. Since, Android Security Architecture, grants permission to perform any type of operation and So Google proclaimed that, “We tried really hard to secure Android. This is definitely a big bug. The reason why we consider it a large security issue is because root access on the device breaks our application sandbox."[1].

To solve this issue, service provider or smart phone marker adopts the new technology to solve this problem called trust zone [1]. Trust zone is the technology has gained wide acceptance and development in recent times. ARM trust zone is a hardware based system virtualization, help in handling third party application and security features in operating system. It consists of two zones called “Normal World” and “Secure World” [2]. The application which requires secure process will enter into secure zone from normal zone. There will be supervisor in the secure zone who will access the data from the normal zone and process it in secured way. The main functionality of the chip (ARM trust zone) will handle memory management unit, input and output guidance of the data, handling cryptographic keys and certificates etc. Since all these functionalities are internally organized, users and developer not aware of it. Android emulator is helpful for designing the business processes functionalities like application, transaction and payment etc whereas upcoming applications (trusted) for secure process cannot be resolved by the android emulator since, any support for software virtualization in it.

In this paper, we propose a design framework for emulated trust zone for android emulator. Our emulated trust zone was designed based on the important attributes used in ARM trust zone design. Furthermore, our designed system also handles the attributes in same way like ARM trust zone. So, it works as a replica for hardware trust zone chip in emulator.

The main contributions of our work can be summarized as:

- Reviewing current practices and theories on implementation of ARM Trust zone.
- Analysis and design the appropriate model by comparing the actual working of trust zone in hardware level with design and idea of emulated one.
- Create a trusted application of our choice matches with design criteria.
- Proposing step-by-step approaches to solve the research tribulations.

2. Problem Statement

In this section, we describe the problem and motivate the need for information to design the emulated trust zone. We are primarily concerned on the emulation of trust zone in virtual emulated world. The main drawback is we cannot focus on single model or design. So, we have to compare different entities like real smart phone trust zone with android emulator functionalities. The practical difficulties in knowing the attributes of ARM trust zone and design & develop the software module matching to it.

The list of attributes used in ARM trust zone [2] need to be replicated:

- Secure memory management
- Monitor mode and supervisor mode
- Interrupts
- User space
- Trusted Application with secure channeling

Table 1. Difference between ARM trust zone and emulated trust zone

	ARM trust zone	**Emulated trust zone**
Secure Memory Management	Created during booting time	Created during execution after SCM call
Monitor mode	NS and S bit value changes according to transit between the worlds	NS and S bit value changes according to transit between the worlds
Interrupts	Operating system takes care of it	Operating system takes care of it
User space	Application start from user space and it enter into kernel space	Same as ARM trust zone functionalities
Trusted Application	Loaded during boot time and it’s static	Loaded during execution time and its dynamic
SCM Call	ARM instructions [8]	Procedure call
Debugging	We cannot debug application since it already compiled	Here, Native C code in JNI layer user” *.so” file
Registers	ARM registers	Variables

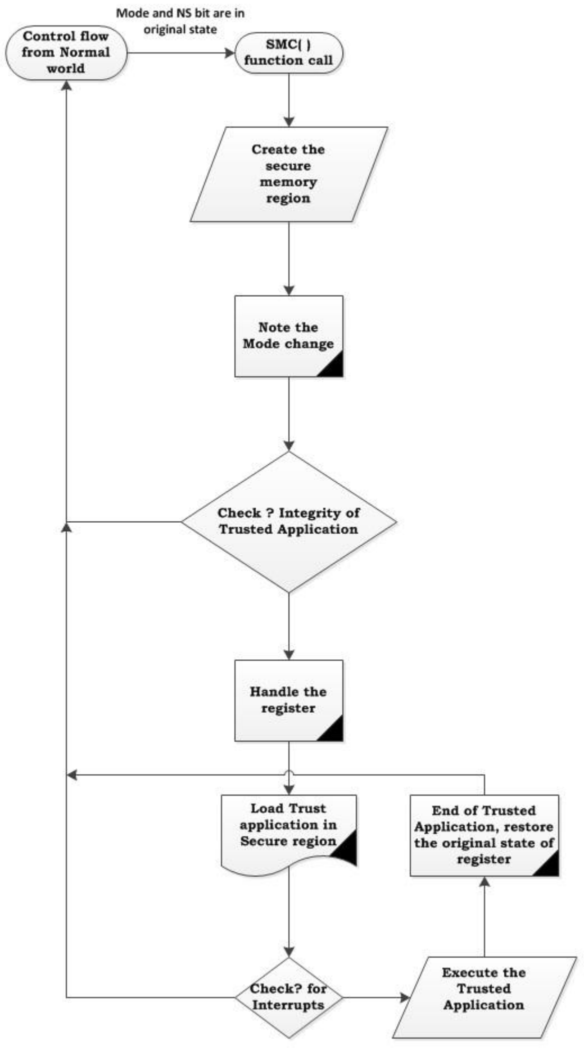
Control flow from Normal world
Mode and NS bit are in original state
SMC( ) function call
Create the secure memory region
Note the Mode change
Check ? Integrity of Trusted Application
Handle the register
Load Trust application in Secure region
End of Trusted Application, restore the original state of register
Check? for Interrupts
Execute the Trusted Application

Fig. 1. Assumed control flow

By considering all these attributes, we can implement emulation in two ways.

- Top level emulation (above driver)
- Low level emulation (below driver)

2.1 Explanations about these two types of emulation

- We continue with the SMC (Secure monitor call) emulation track which gives us a clean/empty secure world that we have to fill with executable code and help to create our own Trusted environment (e.g. Global Platform TEE)[2].
- We create our own TZ driver level, to load trusted application in secure world after SMC call and exchange between the worlds [7].When we try to emulate SMC instruction – we need to replace it with the suitable function (system call) which is targeting the particular address location of the trust application (normal C function call) in the file system.

Problems in Low level emulation:

- Low level emulation is not achievable, due to the compatibility problem between the ARM board (phone) and Intel board(host machine).In real phone there are two different memory unit and accessing point to prevent and guide the execution flow, where as in emulator, single memory strip and single mode operation.
- The trusted code is split into two parts, when loaded by the boot ROM. The first part is regarded as trusted application persistent in secure memory location during the control flow. The other part is initialized or triggered from user space (application). This indirect leads to undefined problem, that is trusted code cannot be unloaded or reloaded [8].
- Trusted application (TA) uses the functionality of the Secure ROM API. Secure ROM API is the main API to be used by trust application (TA) in ARM. So, it is not possible for dynamic trusted application to use defined functionality. So TA should always be static and specifications are loaded in boot time itself.
- Accessing global variable is difficult. No firmware is available to execute or support it.

Problems in Top level emulation:

- Main problem is to create our own execution flow– Android application, Trust zone Driver (TZ Driver), SCM mocking code and trusted application.
- We need to create the fake monitor code i.e. initialization of trusted application and its properties (key, ID, Data, flag and address etc) is mocking the boot ROM specification.
- Handling register functionalities are difficult, since no CP15 register [Appendix] for secure and non secure banking. So, we need to assume and design our own apk layers and TA to perform secure world operation.
- Some verification of trusted application need to done before calling driver codes. This indicates that TA is loaded dynamically and led to problem of in granting permissions to the user.
- Coping TA to specify address (secure memory region) has no meaning in it. Since, we need to bring realistic view of ARM processor.
- No details about secure ROM API: so we create monitor mode as variable or flag status to indicate the user or programmer that, CPU mode has been changed and control flow is switched to secure processing. By reviewing all these entire problems, we prefer, Top level approach is more relevant in this case.

3. Various Designs

By considering these problems with both levels, we suggest top level matches which have feature specifications for replicating the ARM trust zone [5]. Those are:-

- Supervisor design
- Dual memory design
- Static memory design

3.1 Supervisor Design

In this design, we use two instance of kernel layer (i.e.) normal and secure. Supervisor is responsible for the analysing the instruction and then send them into appropriate world. This shows that, supervisor is responsible for context switching between the worlds. The supervisor design is more efficient since it differentiates the two kernels layer and therefore the two worlds. However implementing this solution is more difficult, since one software module will need to understand the already existing and compiled code [5].

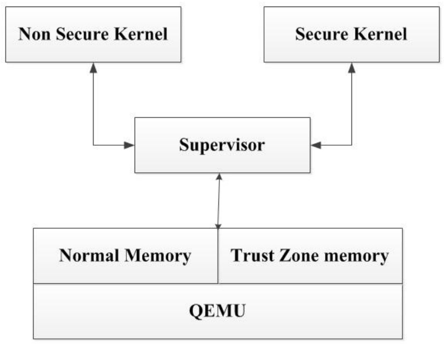

Fig. 2. Supervisor design

3.2 Dual Memory Design

In this design, we create two memory management units to manage each kernel layers to support previous design (supervisor design). As like previous design, here two memory units represent two worlds. The secure memory unit is responsible for the context switching between the worlds. The normal memory unit, will access the part of secure memory unit in order to convey the results between them. So, this indirectly leads to creation of the shared memory region.
This design is not so efficient to isolate the two worlds, even though it solves the supervisor design problem i.e. creation of software module to the stack. Handling memory unit makes this implementation of design more complex [5].

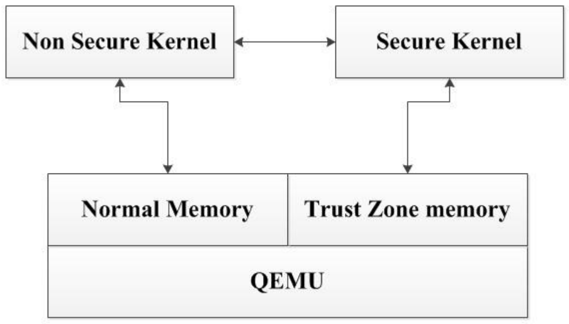

Fig. 3. Dual memory design

3.3 Static Memory Design

By seeing all these complexity in the nature of design, we will stick to one kernel and memory unit. Isolation of secure world can be achieved by creating the static memory in the section of physical memory. The Kernel will monitor each instruction and will forward it according to the appropriate memory region. The kernel is responsible for the context switching. The secure world can access the data from both world and communicate results to normal world without any shared memory concepts [5].
This design is much easier comparatively to other design mentioned above. However, the main problem is handling security features (i.e.) making static memory region has secured one.

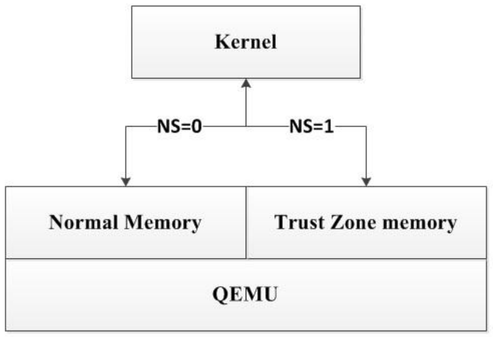

Fig. 4. Static memory design

4. Design Framework

In this section we describe the feasibility abstractions that enable the high level design specification of trust zone in emulator. By clearly viewing the design specification problem, we prefer top level approach with static memory design is more relevant in designing the emulator trust zone in emulator.
Benefits of top level design:

- working as standalone program
- debugging is quite easy
- Navigation from APK->Driver->SCM_CALL->TA is notable.
- We can protect the memory region (secure memory) using mmap methods.

```
/* buffer is temporary variable to
store executable codes */
buffer=mmap(NULL,cmd.Size,PROT_EXEC|PRO
T_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS,1,0);
```

4.1 Design impact of Top level design on attributes of the emulated trust zone:

Secure memory management.
We followed the same principle and design to develop the trust zone features in an emulator. But there is slight modification in the design, when we consider the same approach in emulator. There is no special hardware support for the work flow. Since, in real target there is concept called shared memory between the processor and memory units, but it is missing in emulator. So, we redesign the resource utilization as per emulator accordingly with same NS bit.

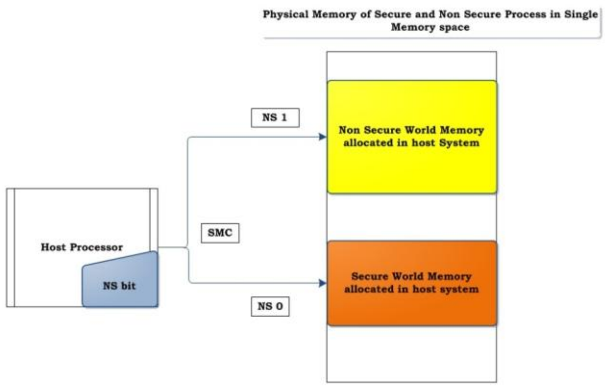
Physical Memory of Secure and Non Secure Process in Single Memory space
NS 1
Non Secure World Memory allocated in host System
SMC
Host Processor
NS bit
Secure World Memory allocated in host system
NS 0

Fig. 5. Secure memory management design for emulated trust zone

Monitor Mode and supervisor mode.

In case of emulator, SMC can be replaced by the function or procedure call.SMC is ARM instruction, which will not work in emulator. SMC function call helps to prevent the non secure state in accessing region of physical memory, since each states operates on own memory address space. Mode changes are captured by the variable values in the programming logic and tracking them are also be done.

Table 2. Bit Value

	Monitor Mode	Supervisor Mode	Non Secure bit	Secure bit
Application in Non secure region	0	1	1	0
Application in Secure region	1	1	0	1

The general purpose registers and processor status register are not blanked between the secure and the non secure states. When execution switches between the non secure and secure states, ARM expects that the values of these registers are switched by a kernel running mostly in monitor mode. Whereas, system coprocessor register are banked between the secure and non secure security states. A banked copy of a register applies only to execution in the appropriate security state.

Interrupts.

Many uses of the security extension can be simplified if the system is designed so that exceptions cannot be taken in monitor mode. Setting bits in the secure configuration register causes one or more of external aborts, IRQs and FIQs to be handled in monitor mode.

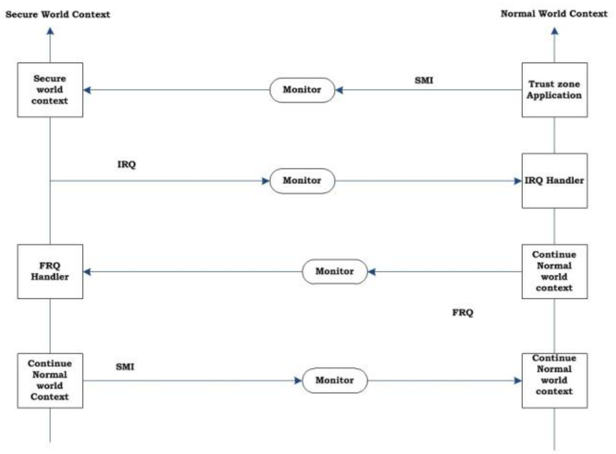

Fig. 6. Interrupt handling

If an exception is taken in monitor mode of non secure state, the Secure Configuration Register (SCR) [Appendix] bit is set to zero [8]. This indicates the operating system that exception occurred. However, if an exception is taken in monitor mode in secure state then, register bit is not set to zero.

User space.

User space is the main feature of the top level design, since the control flow start from it. It helps to delegate operation to an authorized domain. The applications have specific system call to the operating system kernel. This include syscall numbers, interfaces etc. In general, environment for secure world user space application should be simple system call interface to support C run time libraries and compiler tool chains.

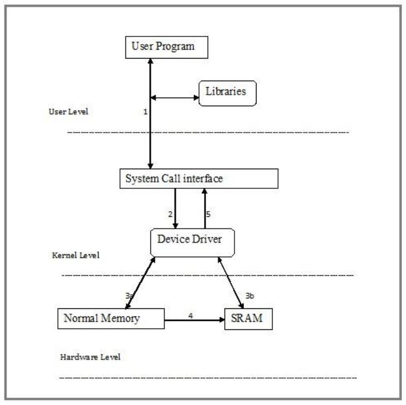

Fig. 7. Control flow from user space

1. Application enter into the user level (android *.apk to system interface)
2. System call leads to corresponding driver file
3. These are things are connected by the JNI- Java native interface.
    3a. initially all process start in normal memory
    3b. whenever the special treatment is required during the execution, control will be directed to secure memory region via driver file when SMC procedure call is made from application.
4. Transfer the block of data from normal memory to secure memory which require the special secure treatment/Process.
5. Once, a process is over, session is closed and returns to normal memory space to further execution.

This process is repeated until process requires special treatment/secure computing

Trusted application.
According to our design, trusted application is the place, where actual business case is introduced to solve and end of the control flow from Top level (apk). For this paper, we constructed the application to establish the secure channel [3] [6] communication between host and external world (for: server of the business provider). So, by replacing the code with desired business logic we can achieve the corresponding significant output.

Why secure channeling?.
In real world application, trust zone works with support of both internal architecture as well as external world. Consider an example of banking transaction, mutual authentication is required by both user and bank server in order to establish the communication between them. This is can be achieved by the "challenge response" mechanism.

Passwords can be reused which may lead to compromise the entire communication. So challenge response is transmitting passwords change each time. Encrypting those passwords with key make the communication more secure.

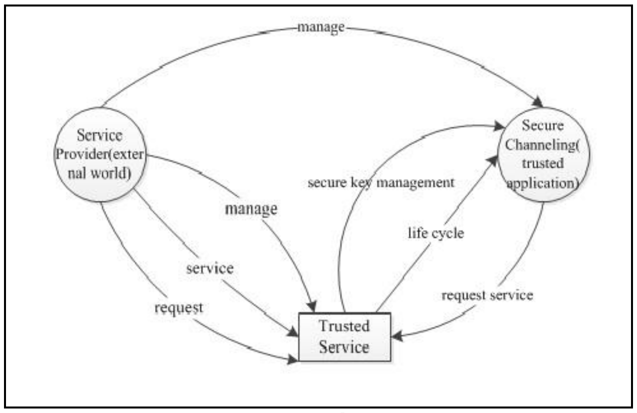

Fig. 8. Trusted services

In smart phone, hardware supports comes with, secure memory region to execute the mechanism (secure channeling) and application created by external service provider. So the user treats black box since he unaware about the mechanism of application. Some of the mechanisms controlled by phone manufacturers are

- Resource of hardware architecture and environment for software execution
- Installing additional application requires permits and assistance
- Billing and usage management are controlled by network operators
- Service management are controlled by service providers so subscribers are not in position to select or change the service

In our case, we use the same principles to design our trusted application. But due to these entire problems mentioned above, we stick to the basic authentication with standard encryption methods to establish the communication between the host and external world to supports our design matches with real world mechanism.

We assume in trusted application mechanism of establishing the secure channel between the mobile and external world is handled. So, trust zone handle the mechanism and input data. Secure channel means mutual authentication between the host (android emulator) and external world. This can be achieved by two way steps.

- Creation of challenge response
- Comparing cryptograms

Initially, we need to create the mutual authentication mechanism, which can be understood by the host and external world. This mechanism is called as challenge response [4].

Random block of 8 bytes should be created by both the host and the external world application called as the host and external challenge. Derivation challenge data of 16 bytes should be formed by the combining host and external challenge [3] [7].

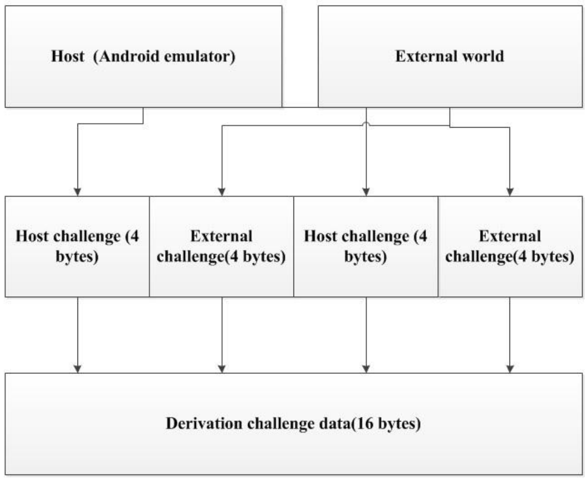
Host (Android emulator)
External world
Host challenge (4 bytes)
External challenge(4 bytes)
Host challenge (4 bytes)
External challenge(4 bytes)
Derivation challenge data(16 bytes)

Fig. 9. Creation of derivation challenge data [3] [7]

Encrypt the derivation challenge data with Static encryption key to form session key. By using the session key, cryptograms are produced. Input data (credentials, pin numbers etc) are combined with the host challenge and send to external world application.

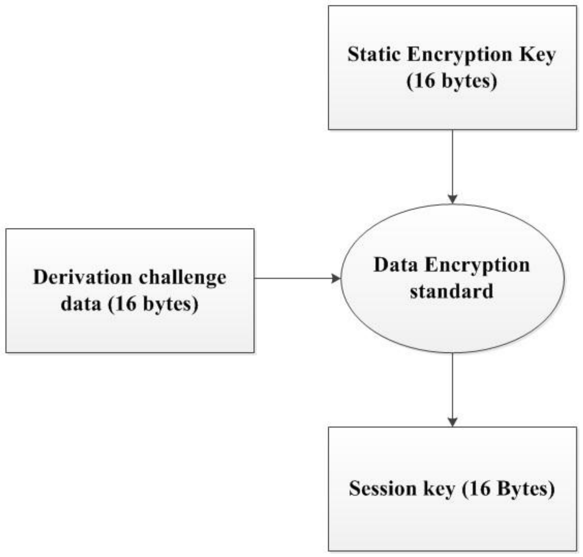
Static Encryption Key (16 bytes)
Derivation challenge data (16 bytes)
Data Encryption standard
Session key (16 Bytes)

Fig. 10. Session key creation

External world application checks host cryptogram and compares it with his own cryptogram (external) generated by the same operation with data from host cryptogram. Secure channel is established if the both the operation leads the same result. So our assumed of communication between host and external world with secure channeling after SCM call will be:

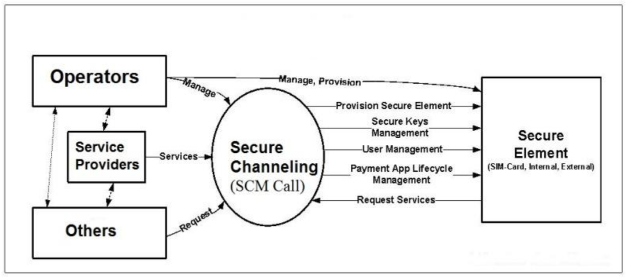

Fig. 11. Communication between host and external world [6]

5. Validating System Model

In this section, we describe our experiments to evaluate and validate design framework. In order to achieve the design of emulated trust zone, we have shown the evidence in analyzing various designs. As mentioned in earlier chapters, Model consists of two layers.

Apk creation or top layer design, defines only certain set of android function call which call the native c procedure which connects the kernel to top layer.

Layer 1: SCM function, actual communicating elements that include driver and procedure for connecting layer 2

- Secure memory creation
- TA creation

Layer 2: Secure channeling, defines the communication between the TA and external application.

5.1 State transition

Considering our example, there according to labelled transition system are 3 stages s0, s1 and s2, where s0 is normal state, s1 is secure state and s2 is external state communication. Process implicitly by calling initial state s0 and follow its successor state S=2T. So the successive states can be defined as successive state(s) = {(s, a, s1)|(s, a, s2) $\in$ T} where T is transition of states. Since SCM call is the initial state of the process and simple "C" procedure for implementation make it straightforward verification algorithms.

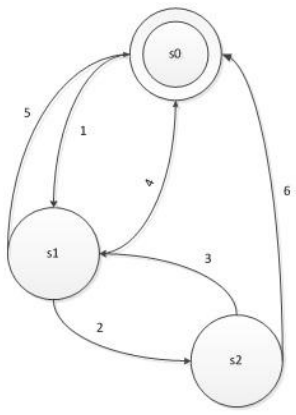

Fig. 12. Transition states

1 & 4: S0 to s1 – transition from top layer APK to SCM call or inturns normal world to secure world.
2 & 3:S1 to s2 – secure channelling communication between host (TA) and external world
5: when interrupt occur in secure processing, control will be revert back to normal world.
6: when interrupt occur in communication between host and external world also, lead back to normal world.

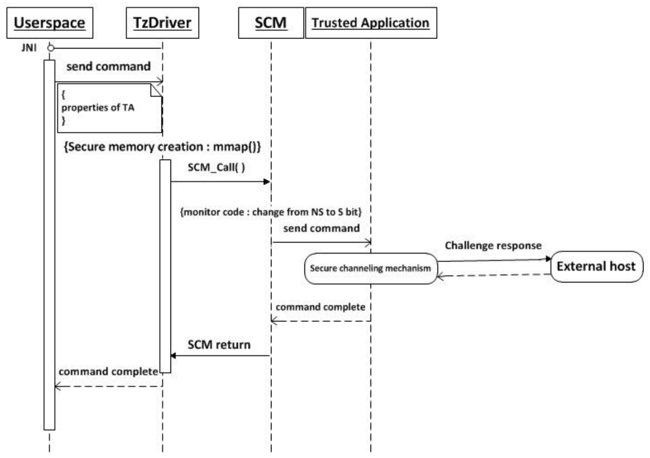

Fig. 13. Control flow from user space to SMC call

5.2 Order of algorithm

Order of the algorithm we used in SCM call is very simple and linear functionalities are used. So the order of the program will O(n2).

Pseudo code with order O(n2):.

```
//CPU_common.h
// we need to define the parameter used
by the SRAM
Typedef unsigned long sram_addr_t; //
this way we can create data type of
SRAM
Sram_addr_t size, offset; // this
parameter will be used by SRAM, to
describe its characteristic
/* In same file, we need to create the
definition for the methods/Functions
used by the QEMU, like allocating the
memory, releasing the memory of SRAM */
Void qemu_sram_alloc (sram_add_t size);
Void qemu_sram_free (sram_addr_t size);
// implementation of this methods can
be viewed in android_arm.c file

//Android_arm.c
Void qemu_sram_alloc (sram_add_t size)
{
/*This will allocate the memory (say
for example 512 Mega bytes) other than
memory create by emulator for normal
function*/
}
Void qemu_sram_free (sram_addr_t size)
{
/* whenever, application is shut down/
Close memory used by the SRAM should be
de allocated */
}

// tee_sram_driver.c
/*this file will be core of the SRAM
feature. This only decide to access the
data in the particular memory location
*/
/* Methods used for the Tee
specification to handle the SRAM
function like read, write, open, close
etc */
Tee_sram_open (); // open the driver
Tee_sram_close (); // close the driver
Tee_sram_read (); // for read operation
Tee_sram_write (); //for close
operation
Tee_sram_session (); // for creating
the session
Tee_sram_reset_session (); // reset the
session
Memrefs_normal_to_sram (); // transfer
of data from normal memory to SRAM
memory space
Memrefs_sram_to_normal (); // transfer
of data from SRAM memory to normal
memory space
```

But in case of secure channeling algorithm, order of the growth will be larger since we are using hashing, crypto functionalities. So, the order will be Ω(n2)

Pseudo code with order Ω(n2):.

```
public boolean establishSecureChannel
() throws Exception {
//host
    byte [] hostChanllenge = new byte[]
{};
//card
    byte[] cardChanllenge = new byte[]
{};
};

//session key creation
sessionKey =
deriveSessionKey1(hostChanllenge,
cardChanllenge, KDC_enc_data);

//Cipher the text
Cipher cipher =
Cipher.getInstance("DES/ECB/NoPadding")
;
```

5.3 Static analysis

Static analysis is very efficient way for finding some interaction (in static memory design), may lead to non determinism state .In simple terms called as overlap finding. It depends on structure of the design. By manual examine of the code functions, overlapping makes the statement divergent.

Table 3. Static analyses

	APK	Driver	SCM	Secure channeling
APK	--	X	X	X
Driver	X	--		X
SCM	X		--	
Secure channeling	X	X		--

5.4 Dynamic analysis.

Dynamic analysis is process of proving functionalities (Static memory design) of application in more logical way. Consider, there are two trusted application A1 and A2 with the functionalities f1 and f2. So, that A1 acquires f1 and A2 have f2.By consider one of the test cases of this project: "Trust zone can access only one process/application at a time".

So, A1f1 = Ø where as A1f1A2f2 ≠ Ø, where Ø is the property of trust zone.
It is easy to understand all this features interact with property. We are considering only properties associated with functionalities like f1 and f2 related to Ø1 and Ø2. So we need more selective approach without violating the proprieties Øi for (f1…fi)Ai.

5.5 Feasibility study.

By considering all these parameters and factors mentioned above, with all the three designs of analysis, we formulate the table to with pros and cons of each design.

Table 4. Evaluation of design based on Attributies

Factor	**Supervisor design**	**Dual memory design**	**Static memory design**
Secure memory management	No separate memory handling for accessing two different kernel layer	Two isolated memory area to handle to two kernel region	Single strip of memory which is divided into normal and secure memory region
Monitor mode	Supervisor will handle the control flow between two kernel region	It work same as Supervisor design, only different is supervisor need to handle control flow between two memory	Supervisor is just a bit variable.
Trusted application	It accessed first in normal kernel and then moved to secure kernel	Here, Application is accessed by normal memory first and then moved to secure memory	Trusted application is copied from normal memory to secure memory by changing the bit value of NS and S bit
SCM_CALL	Supervisor will call for secure processing	Its work same like supervisor design	It just the procedure call to inform application require secure processing
Interrupts	Handled by Operating system	Handled by Operating system	Handled by Operating system
Debugging	Since all process happen in kernel space. No chance of debugging	Since all process happen in kernel space. No chance of debugging	By using NDK, we can debug the user space application but not in kernel space
Register	Internal registers	Internal registers	Variables

Table 5. Evaluation of design based on Functionalities

Factor	Supervisor design	Dual memory design	Static memory design
Complexity	High : Since switching between kernel is not easy	High : memory handling and switching is difficult	Low: since all process happen on top level(user space) switching and handling memory are easy
Reuse	No possibilities of re use of code	No possibilities of re use of code	User space application can be re used. We need to change the kernel driver according to user space program
Implementation	Implementation is not easy since two kernel layer involved	Implementation is not easy since two memory layer involved	Implementation is easy
Security	Security features are difficult to handle here	Protecting the memory is highly difficult	We can use any security standard to encrypt/decrypt the communication between host and external world

By seeing above all, Static memory design would more feasible compare to other two. From various evaluation and implementation criteria, we found out the actual working of ARM Trust zone as well as various ways of implementing replica of trust zone (static memory design) in emulator.

We designed the emulated trust zone in such a way to give justification to the real working mode, even trusted application and secure channeling are also designed to provide support to the design. Finally, we have consolidated artefact in descending order, so that reader can easily follow the flow and idea behind each chapter and design of it.

6 Conclusions

Our aim is to develop a software implementation of trust zone in emulator. To achieve this we have analysed the existing design, understanding the feasibility of implementation of those design in emulator. After exploring alternate design of trust zone, we come to the conclusion that, this research finds the solution to make an architecture design to implement trust zone in android emulator with merely justification to real ARM trust zone design. As the result, various architectural designs of trust zone are discussed and best one is chosen. So, the main goal for paper is drawn.

- Handling Memory segment
- Driver files
- Swapping programs to the specific memory
- Handling system call and CP15 [Appendix] variables
- Sample trusted application – Secure channeling

On overall design, our framework starts with android application, navigate to JNI an interface, to provide high level platform independent experience to the developers. Based on assumption, we implement the small example called “secure channeling” that provide the construction and emulation of trusted application running in secure memory region. SCM call is made in kernel layer to communicate with external host (e.g.: server–client program) to provide mutual authentication mechanism.

- Alternate designs and control flow of framework will explain the overall feasibility for the construction of trust zone in android emulator. In our case, we justify that static memory design is more feasible comparatively with other designs.
- Since, we choose static memory design for our case, so separate memory region is created for secure channeling and context switching is the navigation of control from normal memory to secure memory region and vise versa. Meanwhile other features like interrupts, application handling also done to support context switching.
- Here secure channeling [3] is the process of authenticating the host and external world by using Challenge response. This can be extended by using cryptography standards in order to provide more justification to the design.

Future enhancement & significance.

- By bringing Trust zone (static memory design) into software emulation, have plenty of significance for secure processing application and great contribution to Google android development.
- Environment can be extended to all applications required secure treatment to process and testing

of hacking methods over the applications processed in trust zone.

- This project also helps with application of Trust zone emulation. So, it will avoid hardware limitation on the security of the application for open source world.
- Implementing personalized and complied trusted application – FOTA, Banking application etc.,
- Implementing real ARM SCM call in emulator
- Designing more alternate design for trust zone
- Dependencies on hardware will be reduced
- Help future researcher to work on trust zone and security feature of it.
- Successor of Wallet project of Google

Appendix

Table 6.CP15 Register

Register	Description
C0	Main ID Register (MIDR)
	Cache Type Register (CTR)
	TCM Type Register(TCMTR)
	TLB Type register (TLBTR)
	Multiprocessor Affinity Register (MPIDR)
	Processor Feature Register
	Debug Feature Register (ID_DFR0)
	Auxiliary Feature Register(ID_AFR0)
	Memory Model Feature Register
	Instruction Set Attribute Register
	Cache Size ID Register(CCSIDR)
	Cache Level ID Register(CLIDR)
	Implementation Defined Auxiliary ID register(AIDR)
	Cache Size Selection Register(CSSELR)
C1	System Control Register (SCTLR)
	Implementation Defined Auxiliary Control Register(ACTLR)
	Coprocessor Access Control Register (CPACR)
	Secure Configuration Register (SCR)
	Secure Debug Enable Register(SDER)
	Non Secure Access Control Register(NSACR)
C2	Translation Table Base Register 0(TTBR0)
	Translation Table Base Register 1(TTBR1)
	Translation Table Base Control Register (TTBCR)
C3	Domain Access Control Register(DACR)
C4	Not used
C5	Data Fault Status Register(DFSR)
	Instruction Fault Status Register(IFSR)
	Auxiliary Data and Instruction Fault Status Registers (ADFSR and AIFSR)
C6	Data Fault Address Register(DFAR)
	Instruction Fault Address Register(IFAR)
C7	Cache and Branch Predictor Maintenance Functions
	Virtual Address to Physical Address Translation Operations
	Data and Instruction Barrier Operation
	No Operation (NOP)
C8	TLB Maintenance Operation
C9	cache and TCM Lockdown Register and Performance Monitor
C10	Memory Mapping and TLB Control Register
	Primary Region Remap Register(PRRR)
	Normal Memory Remap Register(NMRR)
C11	Reserved for TCM DMA Register
C12	Security Extension Register
	Vector Base Address Register(VBAR)
	Monitor Vector Base Register(MVBAR)
	Interrupt Status Register(ISR)
C13	Process Context and Thread ID Register
	FCSE Process ID Register(FCSEIDR)
	Context ID Register(CONTEXTIDR)
	Software Thread ID Register
C14	Not used
C15	Implementation Defined Register

Acknowledgments

The alacrity shown by Ms.**Anjani Dubey**, Google Hyderabad & Mr. **Karthikeyan S**, NAL India, has gone long way in improving the form and content of this article.

References

[1] David, Kleidermacher. *Bringing Security to Android based System*. [Online][Cited: June 10, 2013.] http://www.iqmagazineonline.com/current/pdf/Pg56-58_IQ_32-Bringing_Security_to_Android-based_Devices.pdf.

[2] **ARM Security Technology.** *Building a Secure System using TrustZone Technology*. [Online] [Cited: June 13, 2013.] http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[3] **Rajaram, Dinakaran.** *Secure Over the Air (OTA) Management of Mobile Applications.* ICT, KTH Royal insitute of Technology. Stockholm : s.n., 2012.

[4] **C. Newman,A. Menon-Sen, A. Melnikov & N. Williams.** *Salted Challenge Response Authentication Mechanism* (SCRAM). [Online] 2010. [Cited: June 17, 2013.] http://tools.ietf.org/html/rfc5802#page-20.

[5] **Alexander M.. Frisvold, Alex Meyer, Nazmus Sakib, Eric Van Buren.** *Android Security. Iowa State University: s.n.,* 2012.

[6] **Zhao, Hao.** *INTEGRATED SECURITY PLATFORM FOR MOBILE APPLICATIONS.* KTH School of Information and Communication Technology, KTH Royal institute of Technology. Stockholm : s.n., 2011.

[7] Global Platform. [Online] [Cited: June 21, 2013.] http://www.globalplatform.org/showpage.asp?code=specifications.

[8] Winter, Johannes. *Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms.* Institute for Applied Information Processing and Communications (IAIK), Graz, University of Technology. Graz : s.n.

About Authors

Arun Muthu is an Information Security Manager in Feeders Technologies, India founded in 2008. He is responsible for providing security solution to the corporate business and research. He works closely with feeders leaders in driving the growth & expansion of the company's revenue & market. He holds Master degree in Information & Communication System Security from Kungl Tekniska Högskolan (KTH), Sweden 2012. His master degree and his earlier background at Cognizant and Sony Mobile provide him with a solid foundation for research innovation in recent years. His areas of interest are Cryptography, Secure Software Engineering, Privacy, and Trusted Computing.

Dr. Rahim Rahmani is an Associate Professor of Computer Science and system science in University of Stockholm from 2012, focusing on QoS and optimization for management of cloud computing infrastructure and wireless sensor networks, such as OpenFlow. He earned a technical doctorate in adaptive Active Queue Management (AQM) algorithms for access routers in heterogeneous networks. He has served as reviewer and in technical committees of international conferences. He is a member of the editorial board of International Journal of Wireless Networking and Communications.

Dinakaran Rajaram joined Feeders Technologies as Researcher in Information Security as well as IT Security Auditor from April 2013.He obtained his Bachelor of Technology (Information Technology) degree in 2009 from Thiagarajar College of Engineering, Madurai and Master of Science degree in Information & Communication System Security from Kungl Tekniska Högskolan (KTH) in 2012. He is recipient of "Erasmus Mundus" scholarship for his master studies. His researches interests are Security Protocol design, Trusted Computing, Identity Theft, Cryptography and Secure Channeling.