AES Encryption Algorithm using ARM TrustZone Technology

by

Chia-Che Lee

A thesis submitted to the Office College of Engineering at Florida Institute of Technology in partial fulfillment of the requirements for the degree of

Master of Science in
Computer Engineering

Melbourne, Florida
November 2016
We the undersigned committee hereby approve the attached thesis, AES Encryption Algorithm using ARM TrustZone Technology by Chia-Che Lee.

Advisor: Fareena Saqib, Ph.D.
Title: Assistant Professor
Department of Electrical and Computer Engineering

Advisor: Samuel Kozaitis, Ph.D.
Title: Professor and Department Head
Department of Electrical and Computer Engineering

Advisor: Debasis Mitra, Ph.D.
Title: Professor
School of Computing
Abstract

Title: AES Encryption Algorithm using ARM TrustZone Technology

Author: Chia-Che Lee

Advisor: Fareena Saqib PhD.

With the huge impact of IoT devices, the security needs are becoming more important than what it had before. Since embedded systems have limited resources, the traditional security solutions are not quite fitting to IoTs because the huge power consumption and resource. ARM TrustZone Technology is a hardware based security feature using in ARM-based devices with minimum resource taken. TrustZone provides a critical environment to isolate security and non-security tasks within Normal and Security World. Both worlds can be a General Purpose Operating System (GPOS) or a Real-Time Operating System (RTOS) depending on designer has needs, and running simultaneously on the same processor. In general, cases, Normal world will be GPOS such as Linux, Android or iOS, and Secure World will be RTOS, because of the security tasks normally is “Time Critical”. In addition, a Secure Monitor handles the communication between Normal World and Secure World. Secure Monitor provides a set of Application Program Interface (API) as Secure Monitor Call (SMC) to provide secure functionalities for both worlds. The SMC runs the corresponding instructions to control Advanced eXtensible Interface (AXI) interconnects with the hardware security settings. This paper demonstrates an ARM Trustzone firmware implementing on Xilinx Zynq-7000 SoC, which is a tight integration of ARM processor (Processor System, PS) and Programmable Logic (PL).
In addition, the paper is going to explaining details of TrustZone hardware configurations base on SAFE G, an open source TrustZone platform. The goal is analysis ARM TrustZone technology and programing a DEMO application. In this demonstration, I implemented a custom IP as PL to test the correction of functions of TrustZone, and an AES encryption as an example application to run on the system.
Table of Contents

Table of Contents ....................................................................................................... v
List of Figures ........................................................................................................... vii
List of Tables ............................................................................................................. viii
Acknowledgement ..................................................................................................... ix
Chapter 1 Introduction ............................................................................................... 1
  Economic Value in Security Issues ........................................................................ 3
  Security of ARM TrustZone .................................................................................... 4
Chapter 2 Background and Related work .................................................................. 6
  Trusted Platform Module ........................................................................................ 6
  Trusted Execution Environment ............................................................................. 6
  TPM vs TEE ............................................................................................................ 7
  Hardware based Access-Control designs ............................................................. 7
  Background of the ARM TrustZone Technology .................................................... 8
  Zynq - 7000 All Programmable SoC ................................................................. 10
  Support for TrustZone in Zynq-7000 ................................................................... 13
  TrustZone Support in Zynq 7000 Processing System (PS) ................................... 15
  CP15 Register Access Control .............................................................................. 16
  TrustZone Support in Zynq 7000 Programmable Logic (PL) ............................... 18
  Protection from illegal memory access ............................................................... 20
  Briefly View GPOS vs RTOS ............................................................................... 21
  Advanced Encryption Standard (AES) Encryption ................................................ 22
Chapter 3 Experiment steps and Methodology ......................................................... 26
  Implementation of Trusted Execution Environment (TEE) on Zynq 7000 ............. 26
  Safe G and FMP ................................................................................................... 26
  TrustZone Configuration Registers Setting in Zynq-7000 .................................... 28
  Secure Memory Configuration Setting ............................................................... 30
  Booting Sequence on TrustZone ......................................................................... 33
  TrustZone SMC API ............................................................................................ 34
  Hardware Secure Configuration Setting ............................................................ 36
  AES Encryption Execution Flow chat ................................................................... 38
  Booting system steps for Zynq-7000 ................................................................. 40
Chapter 4 Running Result ......................................................................................... 41
  Communication between PS and PL ................................................................. 41
  Communication between Secure World and Normal World ............................... 44
  AES application on TrustZone .......................................................................... 46
Chapter 5 Conclusion ............................................................................................... 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6 Future Work</td>
<td>49</td>
</tr>
<tr>
<td>Physical Unclonable Function (PUF)</td>
<td>49</td>
</tr>
<tr>
<td>References</td>
<td>51</td>
</tr>
<tr>
<td>Appendix</td>
<td>57</td>
</tr>
<tr>
<td>Safe-G Toolchain requirements</td>
<td>57</td>
</tr>
<tr>
<td>Demo files included</td>
<td>58</td>
</tr>
<tr>
<td>Demo files execution steps</td>
<td>58</td>
</tr>
<tr>
<td>AES Demo code for dual-OS Architecture</td>
<td>58</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 — The internet of things was “Born” between 2008 and 2009 [3]. ........2
Figure 2 — attacking an Application on phone [4]..............................................3
Figure 3 — Diagram of ARM TrustZone Technology ........................................9
Figure 4 — Hardware access control on ARM TrustZone. [Source: XILINX]......10
Figure 5 — Zynq – 7000 AP SoC Block Diagram. [Source: XILINX].............11
Figure 6 — the diagram of firmware level execution on TrustZone. ...............16
Figure 7 — Enabling/Disabling Secure Feature at AXI Interconnect. [Source: XILINX]...............................................................19
Figure 8 — An Example of TrustZone memory protection............................21
Figure 9 — The Size of key and block for AES encryption.............................22
Figure 10 — AES round transformation..........................................................23
Figure 11 — AES-SubBytes step...................................................................24
Figure 12 — AES-ShiftRows step.................................................................24
Figure 13 — AES-MixColumns step...............................................................25
Figure 14 — AES-AddRoundKey step...........................................................25
Figure 15 — Architecture of TOPPERS SafeG/FMP. [Source: Safe-G]...........27
Figure 16 — the definition of Secure memory setting [5].................................31
Figure 17 — Secure memory setting in secure monitor...................................32
Figure 18 — TrustZone Boot Sequence. [Source: XILINX]............................33
Figure 19 — Project’s Architecture diagram..................................................36
Figure 20 — VHDL Code of multiplier...........................................................37
Figure 21 — Configuration of AXI_GP0 Secure register settings....................38
Figure 22 — AES encryption’s execution flow. [Source: Sage-G]....................39
Figure 23 — PS-PL Security with REG (SECURITY_FSSW_S0) = 0..............42
Figure 24 — PS-PL Security with REG (SECURITY_FSSW_S0) = 1.............43
Figure 25 — Communication between Secure World and Normal World........45
Figure 26 — Encrypted data stored location..................................................46
Figure 27 — AES application using TrustZone.............................................47
Figure 28 — 1-bit Ring Oscillator PUF.........................................................50
List of Tables

Table 1 — Key features of Zynq – 7000 all programmable SoC ..........................12
Table 2 — Peripherals with Secure and Non-Secure access.................................14
Table 3 — Secure Configuration Register bit Functions ...................................17
Table 4 — TrustZone Module Summary on Zynq – 7000 SoC ..........................28
Table 5 — TrustZone Control Registers Summary on Zynq – 7000 SoC ...........28
Acknowledgement

This thesis could not have been realized without a great support team. I am very grateful to all people from HACEs Lab to give me suggestions on different points of the thesis.

Firstly, I would like to express deep gratitude to my guide to thank Professor Fareena Saqib for choosing me a part of her research group at Florida Institute of Technology, Electrical and Computer Engineering Department. She gave me a wonderful opportunity to work in the area of Hardware Embedded Security and Trust.

I place on record, my sincere gratitude to Ali Shuja Siddiqui for his constant encouragement and advisements and giving me valuable assistances to support for completion of this project. These assistances and advises are really helpful and inspirational.

Finally, I also want to acknowledge my parents and member of my family, who has always supported me for complete the master degree. Without you people, I would not be able to study oversea in the first place.
Chapter 1
Introduction

With the rapid development of computer technology, the large number of usage and popularity of the Internet not only has brought people rapid and huge impact, but also changed the mode of human life. However, when getting the information becomes more and more convenient, it is giving a worrying message for people about their security needed. From Cisco’s Global Mobile Data Traffic Forecast, it presents results that 497 million new connections were added in the 2014 alone with the total number of mobile devices hitting 7.4 billion[1]. This article also presents that since 2007, the market of smartphone and Tablets is growing from $50 to above $400 billion[2]. Looking to the future, Cisco predicts there will be 50 billion Internet of Things (IoT) devices connected to the Internet by 2020[3]. These evidences show that the IoT market is increasing in an incredible speed. For example, cellphone is not just a communications tool from the past. Nowadays it gradually becomes indispensable necessities of life. With the popularity of mobile communication devices, smartphone becomes the hackers and viruses next target. When embedded systems connect to Internet, regardless of the device to device or devices to the cloud, they are in the case of Internet communication with each other, so it is a higher chance that the sensitive information will be stolen, it leads to security requirements become increasingly important to protect user data from malicious software. Since security requirements penetrate into all aspects of embedded systems, Manufacturers and circuit designers are faced with unprecedented challenges.
The security of IoT devices can be divided into multiple parts; these are the security of Internet connection software security and hardware security. Each part of security is crucial in the IoT security. Designers must upgrade all features of security, to guarantee that no data stolen by hackers. Majority attention on security of embedded system in software perspective, now people have more attention on hardware-based security. The object is, software-based protection only prevents software attacks, that means the protection is vulnerable from hardware attack such as read/write and disassemble software to analysis the weakness[4]. However, hardware-based protection can avoid the above attacks. Nevertheless, in various types of attacks, the security protection may only can prevent one type of attack. It is impossible to check all type of attacks. In other words, a security design that protects the wrong resources against an incorrect or incomplete list of attacks can be easily broken. Furthermore, spending long enough time and money, any security design can be broken. the security requirements for a design should not be defined...
as “impossible to bypass” but should be described in value terms: “attack A on asset B should take at least Y days and Z dollars”[5].

Figure 2 — attacking an Application on phone [4].

Figure two shows how the very high value data of critical tasks such as banking application can be stolen by a malicious task using the advantage a security flow in the operation system. Since the security needs, are increasing everyday on embedded devices.

**Economic Value in Security Issues**

Every security issues can be a new business models. Providing servers to give users privacy from cyber-crime become more important nowadays. In addition, not just protections, users want to have freedom of choosing their source of service. Any kind of plagiarism or fraud can be a huge economic cost. The number of cyber-crimes such as stealing credit cards information, plagiarism of corporate environments is increasing because of the different models. “Security will become an issue of
competitive differentiation. Handset devices with inappropriate levels of security will be left on the shelves”[6]

ARM is introducing a 32-bit Cortex-M processors architecture that will bring greater security to the growing numbers of embedded and IoTs(Internet of things) devices[7]. The number of connected devices such as mobile devices and automonies to industrial systems and home appliances continues to grow. In CPU core design, integrating system security extensions is one of the important characteristics of expansion in TrustZone. Under the premise of power consumption, performance and silicon area, it does not have a big impact to entire system. TrustZone aims to prevent the attack surface for hackers in order to satisfy the needs of IoT devices for the next several years[8].

**Security of ARM TrustZone**

The basic idea of TrustZone, has hardware for security boundaries and a minimal core operating system component that is considered trusted, is an excellent idea. In order words, TrustZone is designed to separate and isolate non-trusted resources from trusted hardware, and reduce the attack surface available from sensitive data. The main propose of using TrustZone is providing OS kernel to compromise by hackers, the attacks such as access system sensitive data, hide malicious activities, escalate the privilege of malicious processes, change the OS behavior or simply take control of the system[9]. Vendors can use TrustZone to protect sensitive assents diversely. It’s One of the most essential paradigms in use in computer security today is security by isolation[10]. The different basic system implementations can combine and protect the varied type of the assets by using hardware-based access control security, which is different from general key-encryption systems. By using different type of security can greatly increase the
difficulty of hacking the target’s system. In addition, TrustZone has minimal hardware costs compare to other secure modules. The access control is directly implement in Processor system, no need to add a secure block to protect information.
Chapter 2
Background and Related work

Trusted Platform Module

Trusted Platform Module is a hardware specifically secure crypto processor. Most of PC and laptop motherboards includes TPM ICs for hardware-based security in the past decade. Trusted Computing Group (TCG) introduces the specification of TPMs. Now, the most commended way to defense malware attacks is through antivirus software. These software-level defense techniques have a critical and serious flow, which is antivirus, cannot effectively verify itself. When the malwares gain the same access level as antivirus have, Theses defense software can be simply disable. A computer platform needs a place to verify the software states is still trustworthy. There are two important functional components for TPM –a special register set call Platform Configuration Registers (PCRs) and a cryptographic engine that can execute encryption digital signatures. These registers store the values of state of software on the computer platform. TPM ware designed as a passive chip that only responds to software commands and cost $1 or less to widespread adoption. Moreover, TPM usually is a discrete integrated circuit in a 28-pin thin-shrink package, and it can be attached to other chip on the same motherboard[11][12].

Trusted Execution Environment

Trusted Execution Environment is a concept of hardware-based tasks isolation firmware. The gold of TEE is aiming to provide a smaller operating environment that has enough functionality to secure or provide sensitive service. The services of traditional operating systems provide is becoming larger and more
complex every day, so the task of securing these systems needs is increasingly hard. Moreover, the platforms for processing sensitive information now is moving to personal devices such as smartphones or tablets. These devices consume too much power and the speed of the services are slow. The result is these personal devices need a new simple platform not only can execute sensitive tasks, but also has a enough protection to the system[13].

**TPM vs TEE**

We discussed what are TPM and TEE in the previous sessions; both are made for hardware-based security purpose. What is the difference when we compare TPM and TEE? One is not made to replace another; both have their advantages. TEE works alike a bulletproof safe and TPM is a 128-digit combination lock for the safe. TEE provides a secure execution in a virtualized environment safe and secure boot ensure the system software components are in “trusted” state before system lunching. TPM provides remote attestation to confirm that the system has not been tampered and binding that the only RSA key in the chip is encrypted and sealing to guarantee the data is protected when the system is not in the normal state. However, the solutions that rely on TPM are vulnerable for execution and boot attacks because it is easy to circumvent TPM and override software states. In the another hand, TEE is able to work with Bling and sealing by storing key and protected data in a TEE protected addressable area.

**Hardware based Access-Control designs**

In the previous session explains Trust Execution Environment, the following technologies use for TEE implementation.

1. Intel Software Guard Extensions (SGX).
2. ARM TrustZone

3. Samsung Knox

Intel SGX is Intel Architecture extensions designed using “sandbox”, to increase the software security environment. Intel SGX creates a secure enclave, or it can be called a TEE that protected by the OS which is running. In another word, a secure enclave stores private data in a computation, and the code that operates on it[14][15]. It is a little bit different from ARM TrustZone Technology. ARM TrustZone using two halves; Normal World and Secure World, and communicate with a Secure Monitor via SMC(Secure Monitor Call). Intel SGX will just have one CPU running with multiple secure enclaves. Samsung Know is similar to ARM TrustZone technology. Samsung Know uses TIMA (TrustZone-based Integrity Measurement Architecture) to shield the Linux kernel, and add security boot/trust boot in the bootloader[16][17][18].

**Background of the ARM TrustZone Technology**

The hardware-based embedded security solution is based on “Trust Platform”, or “Trust Execution Environment” concept. That is, excepting user mode and privileged mode, TrustZone introduces Monitor mode. Monitor mode decides whether the operating system is trusted or not. In TrustZone design, it separates the two parallel execution of the virtual operating systems, which are Trusted OS and Non-Trusted OS. Monitor controls the switch functions between T-OS and NT-OS (see Figure1)[19]. In addition, the access control covers the processor, memory and peripherals, where defines in Monitor Mode. That means TrustZone Technology is able to satisfy various demands in general purpose[20].
With TrustZone, Figure 3 shows the crucial components. Trusted OS consider as Secure World, and Non-Trusted OS consider as Normal World. A third Monitor mode take control switching between the normal world and the secure world. The space and privileges of memory address between normal world and secure world is isolated. Secure monitor call (SMC) provides instructions that manage communication between two worlds such as switching worlds, or invoke functions[21]. When the monitor calls the system call to switch worlds, the processor will proceed context switch from currently world to opposite world and freeze execution of the normal world via monitor mode[22].
The ARM TrustZone isolates memory region into two partitions, secure memory that reserved for the secure world and non-secure memory reserved for the non-secure world. The figure 4 shows an example of views that hardware access control on each world. The secure memory can access non-secure memory, but non-secure memory cannot access secure non-secure memory. This technique gives the system is able to store/write sensitive information or execute programs without notifying the normal world. In addition, it also allows individual peripherals to be assigned to the secure world[23].

**Zynq - 7000 All Programmable SoC**

In this project, the platform I am using is Zynq – 7000 SoC. It is an ARM Cortex-A9-based processor (PS) with the hardware programmability of an FPGA and 28 nm Xilinx programmable logic (PL) in a single device. As the Xilinx’s first extensible processing platform (EPP) product, it targets embedded systems by using fixable and high speed of computing performance[24].

TrustZone on Zynq – 7000 All Programmable SoC uses access control flags
on AXI-central inter connect to manage the data isolation. Figure 4 illustrates the functional blocks of the Zynq-7000 AP SoC. The PS and the PL are on separate power domains, enabling the user of these devices to power down the PL for power management if required[25].

Figure 5 — Zynq – 7000 AP SoC Block Diagram. [Source: XILINX]

The Zynq-7000 AP SoC is composed of the following major functional blocks:

- Processing System (PS)
  - Application processor unit (APU)
- Memory interfaces
- I/O peripherals (IOP)
- Interconnect
- Programmable Logic (PL)

The detail features of Zynq – 7000 all programmable SoC show in Table 1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Features</th>
</tr>
</thead>
</table>
| Dual-core ARM® Cortex™-A9 Based Application Processor Unit (APU) | - 2.5 DMIPS/MHz per CPU  
- CPU frequency: Up to 1 GHz  
- Coherent multiprocessor support  
- ARMv7-A architecture  
  - TrustZone® security  
  - Thumb®-2 instruction set  
- Jazelle® RCT execution Environment Architecture  
- NEON™ media-processing engine  
- Single and double precision Vector Floating Point Unit (VFPU)  
- CoreSight™ and Program Trace Macrocell (PTM)  
- Timer and Interrupts  
  - Three watchdog timers  
  - One global timer  
  - Two triple-timer counters |
| Caches | - 32 KB Level 1 4-way set-associative instruction and data caches (independent for each CPU)  
- 512 KB 8-way set-associative Level 2 cache (shared between the CPUs)  
- Byte-parity support |
| On-Chip Memory | - On-chip boot ROM  
- 256 KB on-chip RAM (OCM)  
- Byte-parity support |
### External Memory Interfaces
- Multiprotocol dynamic memory controller
- 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2, or LPDDR2 memories
- ECC support in 16-bit mode
- 1GB of address space using single rank of 8-, 16-, or 32-bit-wide memories
- Static memory interfaces
- 8-bit SRAM data bus with up to 64 MB support
- Parallel NOR flash support
- ONFI1.0 NAND flash support (1-bit ECC)
- 1-bit SPI, 2-bit SPI, 4-bit SPI (quad-SPI), or two quad-SPI (8-bit) serial NOR flash

### Interconnect
- High-bandwidth connectivity within PS and between PS and PL
- ARM AMBA® AXI based
- QoS support on critical masters for latency and bandwidth control

---

**Support for TrustZone In Zynq-7000**

ARM TrustZone Technology is different from other security technique. It protects the memories and peripherals on the Processing system (PS) side, also it controls the security or non-security request to propagate the programmable logic based IP. The Programmable Logic (PL) are tightly integrated block designs and are provided AXI interconnect to be communicate with PS and can be configured the IP as Secure or Non-Secure mode. The system designer must think about adoption of these security features early in the design phase so that the security can be built into the system. The following table shows the secure and Non-Secure accessible parts of Zynq-7000. The systems showing Secure can be accessed only through the secure world, saying both are accessible from secure as well as non-secure (Note: Secure World can access everything on the device in any cases)
Table 2 — Peripherals with Secure and Non-Secure access

<table>
<thead>
<tr>
<th></th>
<th>TrustZone Security</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS7 ARM CPU System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARM A9 Core</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>L1 Cache Controller</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>L1 Cache</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>Memory Management Unit</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>SCU</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>L2 Cache Controller</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>SLCR</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>Triple Time-Counter0</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>Triple Time-Counter1</td>
<td>Configurable</td>
<td>Control using SLCR</td>
</tr>
<tr>
<td>Watch Dog</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>SoC CoreSight Debug</td>
<td>Secure</td>
<td></td>
</tr>
<tr>
<td>OCM</td>
<td>Secure and Non-Secure</td>
<td>256 KB RAM can be divided into 4KB secure domains</td>
</tr>
<tr>
<td>DDR memory</td>
<td>Configurable</td>
<td>IC, GPIO, SPI, Ethernet, SDIO, CAN, USB, and UART, Quad-SPI, NOR</td>
</tr>
</tbody>
</table>

TrustZone Support in Zynq 7000 Processing System (PS)

ARM TrustZone provides NS bit configuration for Secure/Non-Secure mode selection and two virtual OSs and a virtual secure monitor running on the processing system in Zedboard to support the configuration. The Secure Configuration Register (SCR) in coprocessor CP15 defines the NS bit configuration. The configuration is set as zero as the secure mode for both processors by default[26]. For changing the configuration from zero to one (from secure mode to non-secure mode), software in Secure Monitor should be the only one who can modify the NS-bit. Both Operating Systems can enter “Secure Monitor” mode by calling the SMC (Secure Monitor Call), the more details about NS bit will be discussed in the next section. In ARM Cortex-A9 core. Zedboard has Secure/Non-Secure configurable options to all external I/O peripherals and AXI interconnects. Those peripherals include OCM RAM, DMA Controller and interrupts, DDR Controller and All PS interconnects to PL. By default, all hard I/O peripherals and AXI interconnects are set to secure mode. The Boot software can configure all the setting during execution. Figure 6 illustrates the diagram of firmware level execution on TrustZone.
The CP15 is a set of system control coprocessors in ARM Cortex-7 CPU and coprocessor CP15 registers controls the MMU. By controlling the MMU, processor system can configure the secure/non-secure settings to internal/external components. C1 Secure Configuration Register (SCR) in CP15 determines the NS bit that if the program can be executed in secure or non-secure world. All the models except the Secure Monitor of the core can operate in secure or non-secure world. In other words, there are both Secure and Non-Secure User mode and privileged modes. Tables 3 explains the functionalities of C1 Secure Configuration Register (SCR).
Table 3 — Secure Configuration Register bit Functions

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:7]</td>
<td>-</td>
<td>UNP/SBZ.</td>
</tr>
<tr>
<td>[6]</td>
<td>nET</td>
<td>The Early Termination bit is not implemented in ARM1176JZ-S processors. UNP/SBZ.</td>
</tr>
</tbody>
</table>
| [5]   | AW         | Determines if the A bit in the CPSR can be modified when in the Non-secure world:  
0 = Disable modification of the A bit in the CPSR in the Non-secure world, reset value  
1 = Enable modification of the A bit in the CPSR in the Non-secure world. |
| [4]   | FW         | Determines if the F bit in the CPSR can be modified when in the Non-secure world:  
0 = Disable modification of the F bit in the CPSR in the Non-secure world, reset value  
1 = Enable modification of the F bit in the CPSR in the Non-secure world. |
| [3]   | EA         | Determines External Abort behavior for Secure and Non-secure worlds:  
0 = Branch to abort mode on an External Abort exception, reset value  
1 = Branch to Secure Monitor mode on an External Abort exception. |
| [2]   | FIQ        | Determines FIQ behavior for Secure and Non-secure worlds:  
0 = Branch to FIQ mode on an FIQ exception, reset value  
1 = Branch to Secure Monitor mode on an FIQ exception. |
<table>
<thead>
<tr>
<th></th>
<th>IRQ</th>
<th>Determines IRQ behavior for Secure and Non-secure worlds:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 = Branch to IRQ mode on an IRQ exception, reset value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Branch to Secure Monitor mode on an IRQ exception.</td>
</tr>
<tr>
<td></td>
<td>NS bit</td>
<td>Defines the world for the processor:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Secure, reset value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Non-secure.</td>
</tr>
</tbody>
</table>

When the Secure Monitor performs the transformation from one world to the other, processor context must be saved. Context Switch includes register banks, from one world and restore those for the other world. Then it writes the NS bit to change the world operation.

**TrustZone Support In Zynq 7000 Programmable Logic (PL)**

Zynq-7000 AP SoC includes FPGA programmable logic. Designers can program the PL with Xilinx soft IP or with custom IP; these IP cores are typically connected via a memory-mapped AXI interface. PS (Processing system) controls the secure configuration of PL in AXI interconnect level as well as the master/slave IP level.

In AXI interconnect IP supporting for TrustZone Technology, The AXI interconnect IP gives an optional Secure bit checking feature that is used to facilitate support for TrustZone technology in the PL[27]. By default, Secure bit-checking ARPROT [1] bits generated by the master IP in PL and pass the transaction to the slave IP. When the system is booting, designer can decide either enable or disable
the secure bit-checking feature on an AXI interconnect master interface (MI). A DECERR will be issued by AXI interconnect as a response when a master IP connected to it initiates a Non-secure read/write transaction. In this way, the AXI interconnect will not propagate any Non-Secure transactions. Figure seven shows the example of secure option setting on AXI interconnect[28].

![Figure 7 — Enabling/Disabling Secure Feature at AXI Interconnect. [Source: XILINX]](image)

If the Secure option at interconnect master interface M1 is enable, the Non-Secure transaction from Non-Secure Master cannot propagate, and the secure transaction from Secure Master will propagate. In another case, if the secure option at interconnect master interface M2 is disable, both Non-Secure and Secure Master can propagate their transactions to Slave 2.
Protection from Illegal memory access

The commend way of sealing information from systems is using testing and debugging function modules as the starting point of hardware attack. The testing and debugging ports are the most effective physical hardware attack vectors available to a malicious actor. The commend ports for embedded systems usually are Trace and Jtag. If there are loopholes in the system design, attackers can access different modules through the debugging bug. Figure eight shows that TrustZone can protect sensitive information from the attacks through JTAG by using Xilinx SDK debugging tool. Thus, it means secure monitor can identify the illegal memory access and successfully block the request. It can prevent adversaries disassemble programs in order to find the weakness from the system[29][30].
The basic difference between using a GPOS or a RTOS is whether the system is time critical or not. The examples of GPOSs are Windows, Linux or Mac OSX. Moreover, RTOSs are VxWorks, uCos etc. Time critical here means the result’s execution from the system should be predictable. It effects the task scheduling algorithms in the system. The scheduling algorithms of GPOS are various types of algorithms, such as fair-enough, or round robin, expect using priority algorithm.
Priority algorithm is used for RTOSs in the cases for time critical situations. For example, if a system needs to complete a transfer transition in a short time. We will need a RTOS for this problem. The priority algorithm will hurt the performance, which means it is not a fast algorithm. In return, it makes sure the execution time will be the same. In a RTOS, the secure related tasks are the highest priority that will occupy the processor in order to prevent malicious attacks to stall or break the task[31][32].

**Advanced Encryption Standard (AES) Encryption**

This project demonstrates an AES encryption on Trusted OS on TrustZone and runs the program successfully. The AES encryption is a symmetric encryption algorithm, and is used for encrypt electronic data wisely. It replaces Data Encryption Standard (DES) encryption algorithm to become one of the most popular encryption algorithm in the world so far. The standard of Key and Block length is 128 bit, and represented with a matrix (array) of bytes with 4 rows and N columns, $N = \text{key length} / 32[33][34]$.

![Figure 9 — The Size of key and block for AES encryption.](image)

The AES algorithm processes on a two-dimensional array (4 times 4) of bytes called the State. Initially, for the AES round transformation, the first state us the input
plaintext and the final state is the encrypted output. The round transformation mixes the bytes of the State either individually, row-wise, or column-wise by directing the functions Sub-Bytes, Shift-Rows, Mix-Columns, and Add-RoundKey sequentially\[35\]

Figure 10 — AES round transformation.

In the Sub-Bytes step, it will replace values using an 8-bit Rijndael S-Box. The 8-bit Rijndael S-box is a fixed 8-bit lookup table. Each element in the block will be replace with the corresponding value in the S-Box.
In the Shift-Rows step, the value in the block will be shift base on the row location. For example, the first row will not be changed in any case, and the first value in the second row will be shifted in the end of the order... so on so far.

In the case of Mix-Columns, each columns (4 bytes) multiples with a matrix and return a new 4 bytes value. Matrix multiplication composes by multiplying and adding of the entries.
The add-runkey step is that each value XOR with corresponding round-key value. The AES algorithm must execute the entire transformation for 10 times to obscure the relationship between the key and the text value.
Chapter 3
Experiment steps and Methodology

Implementation of Trusted Execution Environment (TEE) on Zynq 7000

Hardware-based TEEs have been widely deployed for over a decade. TI M-Shied and ARM TrustZone are symbolically examples. The concept of TEEs is a combination of features, both software and hardware based, that isolate the execution of tasks. For ARM TrustZone, there are two Trusted Execution Environments support Xilinx Zynq 7000:

1. Sierraware TEE.
2. Nagoya University’s TOPPERS Safe G.

Safe G and FMP

In this project, I am using Safe-G, an open source TrustZone platform from Nagoya University, as a virtual monitor machine in TrustZone. Figure 10 illustrate the Architecture of Safe-G. Safe-G provides a basic secure monitor functions of TrustZone and as a bridge to communicate between Trust OS or Non-Trust OS on Zynq-7000 AP SoC. It designed to execute an RTOS (Real time Operating System) and GPOS (General-Purpose Operating System) on the same hardware platform. Furthermore, it’s flexible with target’s OS platform, which means it can be GPOS/GPOS, GPOS/RTOS and RTOS/RTOS[36]. Figure 15 shows the architecture of Safe-G. Safe-G handles the target’s initialization and secure settings before the Secure OS and the Non-secure OS. By this way, these malicious programs cannot
access any resource when the monitor setting up the target platform. It is essential because on secure concern we do not want the target is interrupted by malicious programs while setting up the secure environment.

TOPPERS/FMP is a high-quality real-time operating system that supports multi-core processors. It is developed within the TOPPERS project and distributed with an open source license. FMP implements a specification that is largely based on the uITRON4.0 interface[37][38].

The main characteristics of FMP are:

- Support for SMP and AMP configurations.
- Kernel and applications are linked in a single monolithic binary.
- Tasks are assigned to processor cores through a configuration file.
- FMP provides runtime system calls for migrating a task to a different core.
- The execution of FMP can be traced and displayed graphically.

Figure 15 — Architecture of TOPPERS SafeG/FMP. [Source: Safe-G]
TrustZone Configuration Registers Setting in Zynq-7000

This section describes the how and where to configure the TrustZone secure settings in Zynq – 7000 SoC. Xilinx Zynq – 7000 SoC defines 22 registers for TrustZone, the based address for TrustZone control registers is showed as Table 4.

Table 4 — TrustZone Module Summary on Zynq – 7000 SoC

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Base Address</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrustZone</td>
<td>TrustZone</td>
<td>F0200000, F8000000</td>
<td>1.0.0</td>
<td>TrustZone Control Registers.</td>
</tr>
</tbody>
</table>

Designers can define the TrustZone secure features by setting up TrustZone Control Registers. Each register has corresponding address and reset value, and these registers are restricted to secure-world access only and are defined and executed in Secure Booting process. The table 5 shows the summary of all the TrustZone Control Registers[39].

Table 5 — TrustZone Control Registers Summary on Zynq – 7000 SoC

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Width</th>
<th>Type</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security2_sdio0</td>
<td>0xE0200008</td>
<td>1</td>
<td>WO</td>
<td>0x00000000</td>
<td>SDIO0 slave security setting.</td>
</tr>
<tr>
<td>security3_sdio1</td>
<td>0xE020000C</td>
<td>1</td>
<td>WO</td>
<td>0x00000000</td>
<td>SDIO1 slave security setting.</td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Type</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>------</td>
<td>------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>security4_qspi</td>
<td>0xE0200010</td>
<td>1</td>
<td>WO 0x00000000</td>
<td>QSPI slave security setting.</td>
<td></td>
</tr>
<tr>
<td>security6_apb_slaves</td>
<td>0xE0200018</td>
<td>15</td>
<td>WO 0x00000000</td>
<td>APB slave security setting.</td>
<td></td>
</tr>
<tr>
<td>security7_smc</td>
<td>0xE020001C</td>
<td>1</td>
<td>WO 0x00000000</td>
<td>SMC slave security setting.</td>
<td></td>
</tr>
<tr>
<td>DMAC_RST_CTRL</td>
<td>0xF800020C</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>DMA Controller SW Reset Control</td>
<td></td>
</tr>
<tr>
<td>TZ_OCM_RAM0</td>
<td>0xF8000400</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>OCM RAM TrustZone Config 0</td>
<td></td>
</tr>
<tr>
<td>TZ_OCM_RAM1</td>
<td>0xF8000404</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>OCM RAM TrustZone Config 1</td>
<td></td>
</tr>
<tr>
<td>TZ_OCM</td>
<td>0xF8000408</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>OCM ROM TrustZone Config</td>
<td></td>
</tr>
<tr>
<td>TZ_DDR_RAM</td>
<td>0xF8000430</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>DDR RAM TrustZone Config</td>
<td></td>
</tr>
<tr>
<td>TZ_DMA_NS</td>
<td>0xF8000440</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>DMAC TrustZone Config</td>
<td></td>
</tr>
<tr>
<td>TZ_DMA_IRQ_NS</td>
<td>0xF8000444</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>DMAC TrustZone Config for Interrupts</td>
<td></td>
</tr>
<tr>
<td>TZ_DMA_PERIPH_NS</td>
<td>0xF8000448</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>DMAC TrustZone Config for Peripherals</td>
<td></td>
</tr>
<tr>
<td>TZ_GEM</td>
<td>0xF8000450</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>Ethernet TrustZone Config</td>
<td></td>
</tr>
<tr>
<td>TZ_SDIO</td>
<td>0xF8000454</td>
<td>32</td>
<td>RW 0x00000000</td>
<td>SDIO TrustZone Config</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Base Address</td>
<td>Size</td>
<td>Access</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>TZ_USB</td>
<td>0xF8000458</td>
<td>32</td>
<td>RW</td>
<td>0x00000000</td>
<td>USB TrustZone Config</td>
</tr>
<tr>
<td>TZ_FPGA_M</td>
<td>0xF8000484</td>
<td>32</td>
<td>RW</td>
<td>0x00000000</td>
<td>FPGA master ports TrustZone Disable</td>
</tr>
<tr>
<td>TZ_FPGA_AFI</td>
<td>0xF8000488</td>
<td>32</td>
<td>RW</td>
<td>0x00000000</td>
<td>FPGA AFI AXI ports TrustZone Disable</td>
</tr>
<tr>
<td>security_fssw_s0</td>
<td>0xF890001C</td>
<td>1</td>
<td>WO</td>
<td>0x00000000</td>
<td>M_AXI_GP0 security setting</td>
</tr>
<tr>
<td>security_fssw_s1</td>
<td>0xF8900020</td>
<td>1</td>
<td>WO</td>
<td>0x00000000</td>
<td>M_AXI_GP1 security setting</td>
</tr>
<tr>
<td>security_apb</td>
<td>0xF8900028</td>
<td>6</td>
<td>WO</td>
<td>0x00000000</td>
<td>APB boot secure ports setting</td>
</tr>
</tbody>
</table>

**Secure Memory Configuration Setting**

This section is going to explain how to set up memory protection on TrustZone. Zedboard has 512Mbyte Memory in the system. The physical address of memory region is:

\[
0x00000000 ~ 0x1fffffff : DDR
\]

Note: 0x00000000 ~ 0x000fffff region is possible to be changed from DDR to OCM.
Figure 16 shows how Xilinx define TrustZone security memory settings are presented into 8 bits. Each bit has 64 MB from 0x00000000 to 0x1ffffff. Each bit of physical address showed as the following

**Bit 0:** 0x00000000 - 0x03fffffff
1: 0x04000000 - 0x07fffffff
2: 0x08000000 - 0x0bfffffff
3: 0x0c000000 - 0x0xffffffff
4: 0x10000000 - 0x13fffffff
5: 0x14000000 - 0x17fffffff
6: 0x18000000 - 0x1bfffffff
7: 0x1c000000 - 0x1fffffff

Where is defined in `/Monitor/target/Zynq/target.c` file.

The figure 17 illustrates a part of codes in secure monitor that preform the secure memory setting. 0xffffffff & ~ (1 << 7); define the memory range 0x1c000000 to 0x1fffffff is secure. Only secure world can access these addresses. The access
requests from non-secure world to these addresses will be defined as a un-registered instruction and the memory region is invisible to Normal World.

```c
static void target_trustzone_init(void)
{
    REG(SECURITY0_SDIO0) = 1;
    REG(SECURITY0_SDIO1) = 1;
    REG(SECURITY0_USPI) = 1;
    REG(SECURITY0_MI0U) = 1;
    REG(SECURITY0_APB_SLAVE) = 0x7FFF; // 1 << 6;
    REG(SECURITY0_UNH) = 1;
    REG(DMAC_RST_CTRL) = 1;
    REG(TZ_DDR_RAM) = 0xFFFFFFFF;
    REG(TZ_DDR_RAM1) = 0xFFFFFFFF;
    REG(TZ_DDR_RAM2) = 0xFFFFFFFF;
    REG(TZ_DDR_RAM3) = 0xFFFFFFFF & ~(1 << 7);
    REG(TZ_DDR_WC) = 1;
    REG(TZ_DDR_LQD) = 0x70;
    REG(TZ_DDR_PERIPHERAL) = 0x8;
    REG(TZ_USB) = 0x8;
    REG(TZ_FPGA0) = 0;
    REG(TZ_FPGA1) = 0;
    // REG(SECURITY_FSSW_SD) = 1;
    // REG(SECURITY_FSSW_SD1) = 1;
    // REG(SECURITY_APD) = 0x1F;
    REG(SCLR_LOCK) = SCLR_UNLOCK_VAL;
    ARM_DATA_MEMORY_BARRIER;
}
```

Figure 17 — Secure memory setting in secure monitor.
Booting Sequence on TrustZone

In Figure 18, solid lines are used to show the boot flow and dotted lines are used to indicate processing transition after the system is running. The shaded blocks are software functional blocks that remain running after the system boots. In the boot sequence, FSBL will first initiate a secure monitor to run the secure settings before the secure and non-secure operation system. After the secure monitor starts, the monitor will execute TrustZone initialization to set up the security features in the system, then jump to the start address of the secure OS. Secure OS defines a set of
events of force transition from non-secure OS to secure monitor. The possible events include SMC instructions, IRQ, FIQ and external Data Abort, and external Prefetch Abort Exceptions.

**TrustZone SMC API**

This section describes how applications in Operating Systems iterate with the secure monitor with security system calls. Secure Monitor provides a set of APIs to both worlds. In the system draft, the system defines eight types of secure monitor system calls for both worlds:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[Static system calls API]</strong></td>
<td></td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__GETID (0)</code></td>
<td>GETID: obtains the ID of a system call by name.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__SETPERM (1)</code></td>
<td>SETPERM: set the permissions for a static or dynamic system call.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__SWITCH (2)</code></td>
<td>SWITCH: initiates a switch to the opposite world.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__SIGNAL (3)</code></td>
<td>SIGNAL: signals an interrupt to the opposite world.</td>
</tr>
<tr>
<td><strong>[Dynamic system calls API]</strong></td>
<td></td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__REGDYN (4)</code></td>
<td>REGDYN: register a dynamic system call.</td>
</tr>
<tr>
<td><strong>[Notifiers API]</strong></td>
<td></td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__REGNOT (5)</code></td>
<td>REGNOT: register a notifier callback.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__READL (6)</code></td>
<td>READL: read specified address.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__WRITEL (7)</code></td>
<td>WRITEL: write specified address.</td>
</tr>
<tr>
<td><code>#define SAFEG_SYSCALL_ID__RESTARNT (8)</code></td>
<td>RESTARNT: restart NT OS</td>
</tr>
</tbody>
</table>
However, in the case for designers who want to register their own function to secure monitor, and be used to both world. They can define a dynamic secure monitor system call for their own purpose. Definition of a system call for registration through safeg_syscall_reg() is:

```c
struct safeg syscall
{
    uint32_t is_t_callable;
    uint32_t is nt_callable;
    uint8_t name[8];
    uint32_t (*function)(uint32_t core_id, uint32_t ns,
                        uint32_t a, uint32_t b, uint32_t c);
}
```

It is a wrapper that covers the actually pointer function (*function). By this way, we can define our own SMC function that can be called by dynamic system calls.
Hardware Secure Configuration Setting

The figure 19 illustrates the project diagram on Zynq 7000 SoC. For the PL IP example, I am using is a 16 to 32 bits multiplier. The example code is the multiplier is showed as Figure 16. The multiplier is written in VHDL and package in a custom IP with an AXI interconnect wrapper. The AXI interconnect is bridge to connect between the PS and the multiplier IP. Processor System Reset IP handles all the reset signal of all the IPs.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multiplier is
  port(
    clk : in std_logic;
    a  : in std_logic_vector(15 downto 0);
    b  : in std_logic_vector(15 downto 0);
    p  : out std_logic_vector(31 downto 0)
  );
end multiplier;

architecture IMP of multiplier is
begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      p <= a * b;
    end if;
  end process;
end IMP;

Figure 20 — VHDL Code of multiplier.

The PL secure setting is handled by modifying corresponding register in secure monitor. Figure 21 shows the detail of configuration registers for M_AXI.GO0. The register defines the request from PS side that either is or is not be propagated.
The AES encryption running in my project is a C language project. In the execution flow, the Non-Secure World proceeds a request to the Secure World via the Secure Monitor. When the Secure World receives the request, the Secure World preform the execution, the execution is invisible from Non-Secure World side. When the Non-Secure World receives the output from the Secure World, then closes the program. Figure 22 shows the details of the execution flow. In this project, AES encryption is given an input with a predictable output, in order to check the correction of the AES encryption execution. Secure Monitor will handle all the operations and communication.
Stage 1: Calling safeg_syscall_regdyn(); to register AES function to Safe G system-call table in Secure Monitor.

Stage 2: Calling safeg_syscall_getid(); to get AES system-call id from system-call table in Secure Monitor.

Stage 3: Calling safeg_syscall_invoke() to invoke AES from trust-OS through Secure Monitor.

Stage 4: Receiving safeg_systemcall_invoke(); executing AES function and sending back the output.

Stage 5: Receiving the output from AES function.

Figure 22 — AES encryption’s execution flow. [Source: Sage-G]
Booting system steps for Zynq-7000

In order to start a system, the following files need to be present on the BOOTROM from the SD card; the files include SafeG and FMP binary files.

1. monitor.bin - it is the binary file containing the software for SafeG secure monitor
2. BOOT.bin - it is the booting file including FSBL, Bitstream, and U-BOOT.
3. fmp_t_sample.bin - it is the FMP secure kernel.
4. fmp_nt_sample.bin - it is the FMP Non-secure kernel.
5. The zynq-7000 SoC sets to SD card boot mode and connect to the computer.

In U-boot mode, enter the following commands to start the secure monitor. The address locations must fit with your secure memory settings in secure monitor. By using fatload command, U-BOOT will place each file to correspond memory location, then use go command to execute the program start at address 0x1c000000.

```bash
mmcinfo;
fatload mmc 0 0x1c000000 monitor.bin;
fatload mmc 0 0x1c100000 fmp_t_sample.bin;
fatload mmc 0 0x8000 fmp_nt_sample.bin;;
go 0x1c000000;
```
Chapter 4
Running Result

Communication between PS and PL

This session presents the communication between PS and PL using TrustZone. In the execution, the Non-Secure World sends request to Program Logic IP block that belongs to Secure World. The PL IP is a simple multiplexer. The Non-Secure World can access the IP directly by accessing the memory address not system call. To show the protection of TrustZone, it shows figures with two different conditions: REG (SECURITY_FSSW_S0) = 1 and 0. When REG (SECURITY_FSSW_S0) is 1, when we expect is the request to access the IP block will succeed, Figure 23 shows the COM3 is Non-Secure World and sends a request to the IP block, and Secure World give response back to Non-Secure. In Figure 24, when REG (SECURITY_FSSW_S0) is 0, we expect Secure Monitor will deny the any request from Non-secure world.
Figure 23 — PS-PL Security with REG (SECURITY_FSSW_S0) = 0
Figure 24 — PS-PL Security with REG (SECURITY_FSSW_S0) = 1
Communication between Secure World and Normal World

This session demonstrates the SMC communication between Secure World and Normal World. In the Demo that shows the Non-secure World calling \texttt{ret = safeg\_syscall\_signal();} to invoke \texttt{syscall\_signal()} in Secure Monitor. The \texttt{syscall\_signal} in Secure Monitor will invoke the \texttt{system\_signal\_handler} in both world. The example shows below:

```c
void safeg\_nt\_signal\_handler(void) {
    static int cnt = 0;
    syslog(LOG\_EMERG, "Non-Trust: received signal \%d\.", cnt++);
}
void safeg\_t\_signal\_handler(void) {
    uint32\_t ret;
    static int cnt = 0;

    syslog(LOG\_EMERG, "Trust: received signal \%d\.", cnt++);
    ret = safeg\_syscall\_signal();
    if (ret != SAFEG\_SYSCALL\_ERROR\_OK) {
        syslog(LOG\_NOTICE, "SafeG syscall ERROR: \%u", ret);
    }
}
```

Figure 25 shows the execution screen shot for the communication between secure world and non-secure world.
Figure 25 — Communication between Secure World and Normal World
**AES application on TrustZone**

The concept of implementing the AES application is similar to the world commutations. The execution flow describes the details of the execution. Figure 27 shows the execution screen of AES application. COM5 is non-secure world that send SMC to secure world, and secure world in COM6 preforms the AES execution then send the resort to non-secure world.

```
Input: {0x6b, 0xc1, 0xbe, 0xe2, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a};

Key: {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};

Output: {0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60, 0xa8, 0x9e, 0xca, 0xf3, 0x24, 0x66, 0xef, 0x97};
```

The Input is the data that we want to encrypt, Key is used for AES encryption key, and output is the encrypted information.

---

**Figure 26 — Encrypted data stored location**
Figure 27 — AES application using TrustZone
In this project, we have deep understanding of ARM TrustZone Technology using Safe-G, an open source TrustZone firmware, and successfully proposed an AES encryption application based on ARM TrustZone technology. Using TrustZone framework provides a higher security for ARM-based embedded systems. It targets to mobile platforms such as Android or IOS operating systems, as well as automotive systems. System designers do not need to rely on traditional key-based encryption to remain the system is trustworthy. TrustZone is a solution that designers have more options to avoid using extremely complex encryptions that will cost huge resource and overheads.
Chapter 6
Future Work

Physical Unclonable Function (PUF)

An on-chip physical unclonable function is a unique challenge-response function, which is providing a random signature/response while the chip is powered-on[40]. Even if the chips are manufactured with the same design and same process, it is infeasible to get the same challenge-response from another same ship with the same conditions. The PUF is hard to predict the responses, which can enhanced for key generation, or creating a true randomness[41][42].

PUF designs in cryptography engine in FPGA has played an important role in security technology progress. The key generation and storage are the indispensable parts for the majority of cryptography implementations. The minimal requirement of security key is that the key must be generate in true randomness. If the security is not made by true randomness that means the most foundation of the cryptographic system is not solid. The adversaries can crack the security key by enumerating huge numbers of the randomness to find the pattern of the key randomness[43]. PUFs can provide quality randomness number with the security key and it cannot be found the pattern by enumerating the randomness. Using PUFs in TrustZone is a decent combination because the TrustZone access control will protect the PUF IP block from the software attacks. By the isolation, the vulnerability in the Non-secure OS will not access the location security key that protected by TrustZone. In this project, we already implement an AES encryption with TrustZone, proving the PUF based security key is the necessary step for the encryption algorithms. Figure 28 shows the RTL design in VHDL. It will be added in the system sooner in the future.
Figure 28 — 1-bit Ring Oscillator PUF
References


Appendix

Safe-G Toolchain requirements

This target has been tested with **Ubuntu 14.04 LTS (64bits)**.

**Vivado version:**
- Vivado 2016.01
- Xilinx_SDK_2016.1

**Safe-G** - Release 1.0
**FMP kernel** - fmp_zynq_gcc-20131203.tar.gz

Applications and libraries that must be installed are:

<table>
<thead>
<tr>
<th>Tool/Library</th>
<th>Packages for Ubuntu</th>
</tr>
</thead>
</table>
| libboost libraries (>1.46) | sudo apt-get install libboost-regex-dev   
|                     | libboost-system-dev \                                                                |
|                     | libboost-filesystem-dev libboost-program-options-dev                                |
| ARM cross-compiler | arm-none-eabi-gcc 4.6.2 or 4.6.3                                                   |
|                    | **Note:** Other compilers or versions have not been tested and may not work.        |
| Other packages     | sudo apt-get install build-essential libstdl-dev                                     |
Demo files included

U-boot.bin - the Universal Boot Loader.

Monitor.bin - SafeG monitor binary file.

Fmp_t_sample.bin - FMP kernel binary file that runs on Trust side (T-FMP).

Fmp_nt_sample.bin - FMP kernel binary file that runs on Non-Trust side (NT-FMP).

Demo files execution steps

Run the following command after the u-boot execution.

```
mmcinfo
fatload mmc 0 0x1c000000 monitor.bin
fatload mmc 0 0x1c100000 fmp_t_sample.bin
fatload mmc 0 0x8000 fmp_nt_sample.bin
go 0x1c000000
```

AES Demo code for dual-OS Architecture

```c
#include <kernel.h>
#include <t_syslog.h>
#include <t_stdlib.h>
#include <sil.h>
#include "syssvc/serial.h"
#include "syssvc/syslog.h"
#include "kernel_cfg.h"
#include "sample1.h"
#include "syscalls_api.h"
#include "utils.h"
/
* AES
```
/*
#define TOPPERS_SAFEG_SECURE
#include "aes.h"
#endif
char * enc_function_name ="ECBen";
Inline void
svc_perror(const char *file, int_t line, const char *expr, ER ercd)
{
    if (ercd < 0) {
        t_perror(LOG_ERROR, file, line, expr, ercd);
    }
}
#define SVC_PERROR(expr) svc_perror(__FILE__, __LINE__, #expr, (expr))
#endif /* TOPPERS_SAFEG_SECURE */
#define TOPPERS_SAFEG_SECURE
void btask(intptr_t exinf)
{
    uint32_t ret;
    syslog(LOG_NOTICE, "BTASK Starts");
    while(1) {
        syslog(LOG_NOTICE, "safeg_syscall_switch()");
        ret = safeg_syscall_switch();
        if (ret != SAFEG_SYSCALL_ERROR_OK) {
            syslog(LOG_ERROR, "SafeG ERROR: %u", ret);
            break;
        }
    }
}
#endif /* TOPPERS_SAFEG_SECURE */
#endif TOPPERS_SAFEG_SECURE
uint32_t ECBen(const uint32_t core_id,const uint32_t ns, const uint32_t addr_in,
    const uint32_t addr_key, uint32_t  addr_buffer){
    uint8_t * in = (uint8_t *)addr_in;
uint8_t * key = (uint8_t *) addr_key;
uint8_t * buffer = (uint8_t *) addr_buffer;

ECBenc(in, key, buffer);
return SAFEG_OK(SAFEG_SYSCALL_RET__DONT_SWITCH);
}

struct safeg_syscall ecb_call;

void define_ECBen_syscall()
{
    ecb_call.is_t_callable = 1;
    ecb_call.is_nt_callable = 1;
    string_copy(ecb_call.name, (uint8_t *) enc_function_name, 7);
    ecb_call.function = (void *) ECBen;
}

// syslog(LOG_EMERG, "Non-Trust: received signal %d.", cnt++);
#endif /* !defined(TOPPERS_SAFEG_SECURE) */

/*
 * Main task
 */
void main_task(intptr_t exinf)
{
    ER_UINT ercd;
    uint32_t ret;
    uint32_t id;
    char tmp[10];

    SVC_PERROR(syslog_msk_log(0, LOG_UPTO(LOG_DEBUG)));
    syslog(LOG_NOTICE, "Simple starts (exinf = %d).", (int_t) exinf);

    ercd = serial_opn_por(TASK_PORTID_G_SYSLOG);
    if (ercd < 0 && MERCD(ercd) != E_OBJ) {
        syslog(LOG_ERROR, "%s (%d) reported by `serial_opn_por'.",
               itron_strerror(ercd), SERCD(ercd));
    }
    SVC_PERROR(serial_ctl_por(TASK_PORTID_G_SYSLOG,}
#ifdef TOPPERS_SAFEG_SECURE
    syslog(LOG_NOTICE, "Hello T-FMP");
syslog(LOG_NOTICE, "running AES Demo");
syslog(LOG NOTICE, "Calling safeg_syscall_regdyn(&ecb_call,&id)" );
define_ECBen_syscall();
    ret = safeg_syscall_regdyn(&ecb_call,&id);
#endif
    syslog(LOG_NOTICE, "Hello NT-FMP");
    while(1) {
        ret = safeg_syscall_getid(enc_function_name, &id);
        uint32_to_string(id, tmp, 10);
        syslog(LOG_NOTICE, "ECB is: %u", tmp);
        uint8_t key[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
                         0xa8, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
        uint8_t in[] = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a};
        uint8_t out[] = {0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60, 0xa8, 0x9e, 0xca, 0xf3, 0x24, 0x66, 0xef, 0x97};
        uint8_t buffer[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        ret = safeg_syscall_invoke(id, (uint32_t)&in, (uint32_t)&key,
                               (uint32_t)&buffer);
        syslog(LOG_NOTICE, "expect ouput:");
        syslog(LOG_NOTICE, "0x3a, 0xd7, 0x7b, 0xb4");
        syslog(LOG_NOTICE, "0x0d, 0x7a, 0x36, 0x60");
        syslog(LOG_NOTICE, "0xa8, 0x9e, 0xca, 0xf3");
        syslog(LOG_NOTICE, "0x24, 0x66, 0xef, 0x97");
        // I dont use for-loop to print output because somehow it wont show
        the right values of the array.
        syslog(LOG_NOTICE, "output buffer:");
        syslog(LOG_NOTICE, "0x%02x,0x%02x,0x%02x,0x%02x", buffer[0],buffer[1],buffer[2],buffer[3]);
        syslog(LOG_NOTICE, "0x%02x,0x%02x,0x%02x,0x%02x", buffer[4],buffer[5],buffer[6],buffer[7]);
        syslog(LOG_NOTICE, "0x%02x,0x%02x,0x%02x,0x%02x", buffer[8],buffer[9],buffer[10],buffer[11]);
syslog(LOG_NOTICE, "0x%02x,0x%02x,0x%02x,0x%02x",
buffer[12],buffer[13],buffer[14],buffer[15]);
  //ret = safeg_syscall_signal();
  syslog(LOG_NOTICE, "AES Demo Done");
  if (ret != SAFEG_SYSCALL_ERROR_OK)
  {
    syslog(LOG_NOTICE, "SafeG syscall ERROR: %u", ret);
  }
  //  dly_tsk(500000);
  // }
#else /* TOPPERS_SAFEG_SECURE */

  syslog(LOG_NOTICE, "Main goes to sleep.");
  SVC_PERROR(slp_tsk());
  SVC_PERROR(ext_ker());
  assert(0);
}

#endif /* TOPPERS_SAFEG_SECURE */