

1-1-2013

Hardware Assisted Security Platform

Mir Ahsan
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>



Part of the [Computer Engineering Commons](#)

Recommended Citation

Ahsan, Mir, "Hardware Assisted Security Platform" (2013). *Theses and dissertations*. Paper 1928.

This Major Research Paper is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

HARDWARE ASSISTED SECURITY PLATFORM

by

Mir Ahsan

BASc, University of Toronto, 2009

A project
presented to Ryerson University
in fulfillment of the
thesis requirement for the degree of
Master of Engineering
in
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2013

©Mir Ahsan 2013

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this MRP. This is a true copy of the MRP, including any required final revisions.

I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my MRP may be made electronically available to the public.

Hardware Assisted Security Platform

Master of Engineering, 2013

Mir Ahsan

Electrical and Computer Engineering

Ryerson University

Abstract

Embedded systems are often used to monitor and control various dynamic and complex applications. However, with greater accessibility and added features on many embedded systems, more and more systems are being subject to sophisticated and new types of attacks. As a result, the security aspect of embedded systems has become critical design step. TrustZone has become a popular choice for security design solution in systems where resources such as processing power, battery are limited. In TrustZone, two virtual processors called "secure world" and "normal world" run on the same core in a time sliced manner. These worlds have partitioned hardware and software resources, with different modes of operation, isolated memory regions and interrupts. In this paper, the hardware and software architecture of TrustZone is analyzed from the perspective of embedded system security design. Then a mobile-ticketing system based on TrustZone is presented which incorporates standard cryptographic engineering design practices to demonstrate the feasibility and effectiveness of such system. The ticketing system is then simulated and security threat analysis is performed in terms known vulnerabilities such as Buffer Overflow, Static and dynamic code/data tampering, Return Oriented Programming (ROP) exploits, and Man-in-the middle attacks. After evaluating the analysis results with various open source vulnerability analysis tools, it is conclusive that the system design is an effective solution particularly for embedded systems.

Acknowledgements

First, I would like to thank my supervising professor, Dr. K. Raahemifar. His dedication and guidance in both technical and non-technical areas made for this project to be completed. I appreciate his vast knowledge and skill in the areas of research.

I must thank my family for their never ending support, especially my wife, Afsana, for her patience, encouragement and help.

Table of Contents

AUTHOR'S DECLARATION	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	vii
List of Tables	viii
Chapter 1 Introduction.....	9
Chapter 2 Background and Related Work.....	11
2.1 Embedded System Security Requirements.....	11
2.2 Embedded System Attack Classification.....	12
2.2.1 Types of Attackers.....	12
Class I (Clever outsiders):	12
Class II (Insider attack):	13
Class III (funded organizations):	13
2.2.2 Types of Attacks.....	13
2.3 Mitigating Embedded System Attacks	14
2.4 Existing Security Solutions	15
2.4.1 External Hardware Security Module	15
2.4.2 Internal Hardware Security Module	17
2.4.3 Software Virtualization	17
2.4.4 Software Access Control	18
Chapter 3 Cryptographic Theory	20
3.1 Cryptographic Tools	20
3.1.1 Symmetric Ciphers	20
3.1.2 Asymmetric Algorithm.....	21
3.1.3 Cryptographic Hash.....	21
3.1.4 Digital Signatures	22
Chapter 4 TrustZone Architecture.....	24
4.1 Trustzone Hardware Architecture	24
4.1.1 System and bus structure	24
4.1.2 Register Space	25

4.1.3 World Switching	27
4.1.4 Memory and Cache	28
4.1.5 Interrupts	30
4.1.6 Peripherals.....	31
4.2 Trustzone Software Architecture	33
4.3 Benefits of Trustzone.....	34
Chapter 5 Example Application: Mobile Movie Ticket Application.....	35
5.1 System Requirements.....	35
5.2 System Block Diagram	36
5.3 Operating System and Cryptographic Processor.....	37
5.4 Normal World User Application.....	37
5.4.1 Tamper Resistance	38
5.4.2 Data confidentiality.....	40
5.4.3 Secure World Kernel API	41
5.5 Key-exchange and software flow.....	41
Chapter 6 Simulation Results.....	44
Figure 24: System Canvas Simulation	44
Figure 25: Schematic of VE with.....	45
Figures 26-28 show screenshots of how Linux is loaded on this platform.	45
6.1 Security Threat Analysis.....	47
6.1.1 Buffer Overflow	47
6.1.2 Static Code tampering.....	48
6.1.3 Dynamic Code tampering	48
6.1.4 Return Oriented Programming (ROP) Exploitation.....	48
6.1.5 Man in the Middle Attack	49
Chapter 7 Conclusion.....	50
Appendix A Bibliography.....	51
Appendix B Simulation Environment.....	54
Appendix C TrustZone API.....	57

List of Figures

Figure 1: Embedded System attack types	13
Figure 2: Power analysis of AES Key processing	14
Figure 3: Attack prevention generic strategy.....	15
Figure 4 Symmetric Cipher	20
Figure 5 Asymmetric Encryption	21
Figure 6: Cryptographic Hash	22
Figure 7: Tampered Message Hash	22
Figure 8a: Digital Signature: Signing data	23
Figure 8b: Verifying signatures	23
Figure 9: Typical system block diagram with share resources.....	25
Figure 10: Arm register specificaiton.....	26
Figure 11: CP15 register contains the NS bit set according to mode of operation .	27
Figure 12: Secure world, normal world and monitor mode.....	27
Figure 13: Cache and TLBs.....	29
Figure 14: TrustZone memory access example	30
Figure 15: Interrupt propagation between worlds	31
Figure 16: Trustzone and secured peripherals.....	32
Figure 17: Trustzone Software Architecture	33
Figure 18: System Block Diagram	37
Figure 19: CRC comparison code	38
Figure 20: CRC check instruction level.....	38
Figure 21: Check sum function pointer table	39
Figure 22: Actual function pointer table for CRC 0x09A1DE9F	40
Figure 23: Key exchange and software flow	43
Figure 24: System Canvas for simulation	44
Figure 25: Schematic of VE with Corex A15 processor	45

List of Tables

Table 1: Advantages and disadvantages of external hardware security module.....	16
Table 2: Advantages and disadvantages of Internal Hardwar Security Module	17
Table 3: Advantages and disadvantages of security through software virtualization ...	18
Table 4: Advantages and disadvantages of software access control	19
Table 5: Arm Processor Modes	25
Table 6: Notations	41

Chapter 1

Introduction

Embedded systems have become integral part of our society over the last decade. In today's world, embedded systems are used to monitor and control various dynamic and complex applications, ranging from non-safety-critical systems such as cellular phones, media players, and televisions, to safety-critical systems such as automobiles, airplanes, and medical devices. The critical nature of these applications, especially in military, medical and transportation industries imply that these systems must be highly reliable as well as secured. Thus security of embedded systems is emerging as a new dimension in modern embedded systems design.

With increasing number of everyday embedded devices, more and more systems are being subject to sophisticated and new types of attacks yielding to higher number break-ins. For example, the iPhone was broken into only days after its public release [1]. The cost of such security breach can be critical depending on the nature of the system. For example, it was estimated that the "I Love You" virus caused nearly one billion dollars in lost revenues worldwide [1]. With an increasing proliferation of such attacks, it is not surprising that a large number of users in the mobile commerce world (nearly 52% of cell phone users and 47% of PDA users, according to a survey by Forrester Research [2]) feel that security is the single largest concern preventing the successful deployment of next-generation mobile services.

While security protocols and cryptographic algorithms address security considerations from a functional perspective, it is unrealistic to assume that attackers will attempt to directly take on the computational complexity of breaking the cryptographic primitives employed in security mechanisms [3]. Functional cryptosystems often do not provide complete system level security as explained by Schneier in [16]. As a result, the security model should be considered from a system point-of-view. However, many embedded systems are constrained by the execution environment and resource limitations such as low battery life, memory

constraints, and lower computational power. For such systems, security is moving from a function centric perspective into a system architecture (hardware/software) design issue. In this paper, first the security requirements for embedded systems are outlined. Then attack patterns are classified and various design solutions such as external hardware and software, virtualization and virtual processors are presented. Then a virtual processor architecture solution called “TrustZone” is described in detail from a hardware and software design view. At the end, a secured mobile-ticketing system designed and implemented based on TrustZone architecture is described.

Chapter 2

Background and Related Work

2.1 Embedded System Security Requirements

Security has traditionally been an active area of research in Desktop and network computing. A lot of network security protocols, such as IPSec and SSL, have been defined to achieve authentication between communicating entities, and ensure the confidentiality and integrity of communicated data [5].

Since the nature of embedded systems varies widely based on the application requirements, the security requirements also vary. In typical systems available today, more than one aspect of security needs to be considered as complex systems may comprise of individual modules with different security requirements. For example, consider a smartphone that is capable of wireless voice, multimedia, and data communications. Security requirements may varies with the viewpoint of the manufacturer of a particular component inside the cell phone (*e.g.*, baseband processor), the cell phone manufacturer, the cellular service provider, the content provider, and the end user of the cell phone. The end user's primary concern may be the security of personal data stored and communicated by the cell phone, while the content provider's primary concern may be DRM related protection of multimedia content delivered to the cell phone, and the cell phone manufacturer might additionally be concerned with the secrecy of proprietary firmware that resides within the cell phone. As mentioned earlier, in this paper, an example mobile ticketing application is explored. The application performs a simple task of letting users browse through and pick a movie, and then stores the ticket issued by the movie operator company in the device. The task at hand is to ensure security of this transaction. There are six security requirements that any embedded system must mitigate:

1. Data confidentiality: Embedded systems must protect sensitive information from undesired eavesdroppers in the communication channel between two modules within a system or between different parties.

2. Data integrity: Embedded systems must ensure that user data stored on the device has not been changed illegitimately
3. Authentication: Embedded systems must validate that the data received is actually sent from the intended and correct sender.
4. Provide content Security: Embedded systems must provide full content security under Digital Rights Management (DRM) protocol to ensure that the data processed from media cannot be copied or altered.
5. Secure Storage: Embedded systems must provide a secure storage for critical and sensitive information such as various keys so that these secrets are not leaked out for the full duration of the device's life time and erased afterwards.
6. Tamper Resistance: Embedded systems must provide tamperproof and may be detection mechanism even when the device can be physically or logically probed.

As stated earlier, the system requirements vary according to the functionality and the typical operation environment. However the requirements stated above cover most of the modern devices today.

2.2 Embedded System Attack Classification

Attacks on embedded systems vary on the type and method of execution primarily based on the nature and availability of the target device. Attackers typically rely on exploiting security vulnerabilities in the hardware and software components. Even though, each attack is different in terms of the methodology, the attackers can be categorized into three classes as described below.

2.2.1 Types of Attackers

Class I (Clever outsiders): This type of attack is launched by individuals who are intelligent but often do not have sufficient information about the systems. Usually these attackers try to take advantage of existing flaw or vulnerability of a system instead of creating one.

Class II (Insider attack): May be launched by defected employee who has specialized technical education and experience. They have varying degrees of understanding of parts of the system but not full access to the system.

Class III (funded organizations): Attacks launched by teams of specialists with related and complementary skills backed by great funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks. Some well-known organizations today are “Anonymous”, and “Masters of Deception”.

2.2.2 Types of Attacks

Ravi et al categorizes attacks on embedded systems into three areas based on the impact level in [3] as shown in figure 1 below:

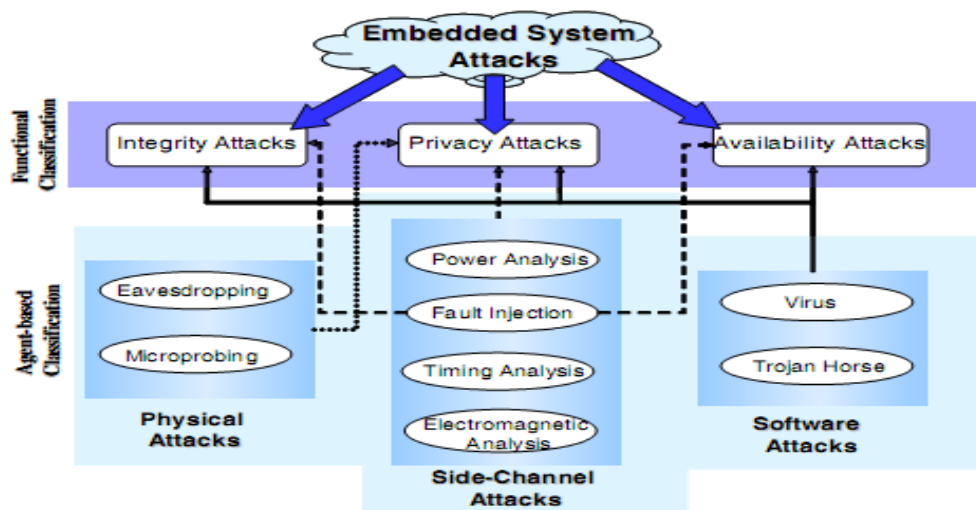


Figure 1: Embedded System attack types

1. Privacy attacks: The objective of these attacks is to gain knowledge of sensitive information stored, communicated, or processed in an embedded system. A common example is the Man-in-the-middle attack, Meet-in-the-middle attack, and Sniffing using software approach. On the hardware level, common methods used by the attackers include power analysis, fault injection and timing analysis to reveal patterns or faults for a window of fault. For example, power analysis can show patterns when

a certain cryptographic algorithm is executed. Figure 2 below shows the power analysis of when AES key is processed, thus leading to exposure of the key.

2. Integrity attacks: These attacks attempt to change data or code associated with an embedded system. Eavesdropping and data manipulation techniques are most common types of attacks using key manipulation and key injection mechanisms.
3. Availability attacks: These attacks disrupt the normal functioning of the system by misusing and suffocating system resources so that normal operation cannot continue. Examples of such attacks are Denial of Service (DoS) attack and Replay attacks.

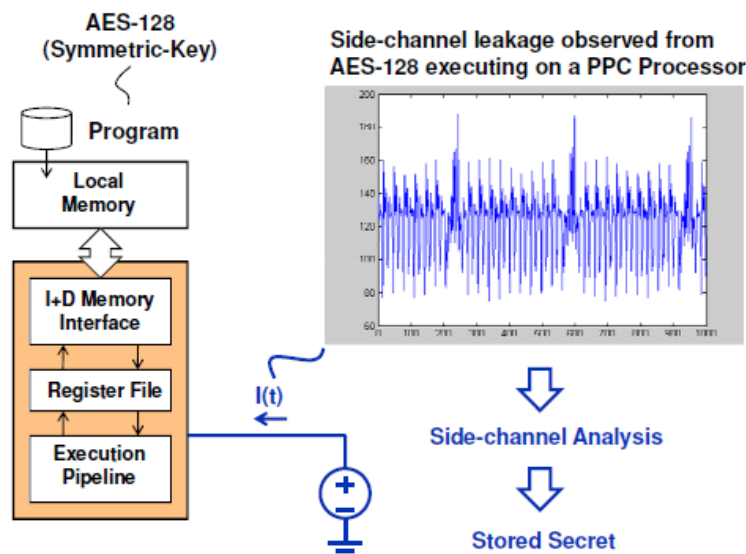


Figure 2: Power analysis of AES Key processing

2.3 Mitigating Embedded System Attacks

To mitigate or counter the attacks described in the previous section, a generic security model can be developed. First line of defense is known as Attack Prevention. These are measures which are in place to prevent the attacker from initiating an attack. Anti-tampering and code obfuscation are examples of techniques used in attack prevention. Then in the next step, there must be mechanism to detect that system configuration or code has been altered. Check sum validations or run-time integrity checks are commonly used in this phase. Lastly, the attacked program or hardware must

take measures to avoid execution or exposure of sensitive information upon detection of attacks, and recover if possible.

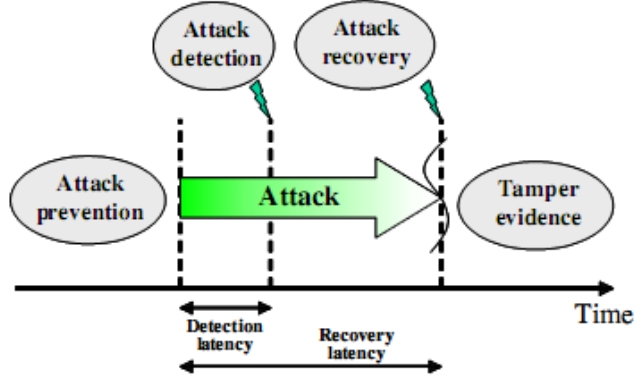


Figure 3: Attack prevention generic security model

When considering full system security solutions, there are a number of common design practices deployed in embedded platforms today. In the next section, four such solutions are analyzed.

2.4 Existing Security Solutions

A number of solutions have been proposed in [4, 10, 11, and 17] using external, internal hardware and software techniques. This section summarizes the pros and cons of some the existing solutions.

2.4.1 External Hardware Security Module

External hardware security module based systems have a dedicated hardware block to perform security related functionalities. This hardware is typically placed outside the Main SoC and have predefined interface to communicate and execute services from the main processor core [8]. A popular example of such solution is the SIM card in a mobile handset, or a smartcard in a satellite receiver, or dedicated hardware block in the GPU to perform DRM operations. Table 1 below highlights the key advantages and disadvantages of external security based hardware designs.

Table 1: Advantages and disadvantages of external hardware security module

Advantages	Disadvantages
<ul style="list-style-type: none"> ✓ The security assets such as keys, algorithms are isolated in a dedicated hardware block ✓ Isolation techniques provide high levels of tamper resistance and physical security ✓ Only authenticated applications can be run on the secured hardware block 	<ul style="list-style-type: none"> ✓ Requires more silicon area, increases power consumption, and cost and reduces performance of the device. ✓ These hardware blocks typically provide a platform for processing and secure storage functions only. An interface is required to the host processor, which adds to design effort. Also, this interface is vulnerable to attacks. For example, a user's entry of a Personal Identification Number (PIN) must be managed by the less secure software outside of the smartcard, making it vulnerable to attack, even though there is SIM hardware.

2.4.2 Internal Hardware Security Module

The security block or dedicated hardware is located inside the main SoC.

Table 2: Advantages and disadvantages of Internal Hardware Security Module

Advantages	Disadvantages
✓ There is significant cost reduction in terms of hardware design, and silicon costs compared to the External Hardware Security Module	<div>✓ There is typically another separate physical security processor, which is less powerful than the main applications processor, and also consumes significant silicon area. Additionally, communication between the two processors requires the use of external memory.</div> <div>✓ Since the security processor is inside the SoC, significant design and test effort are needed.</div>

2.4.3 Software Virtualization

Virtualization is a software security mechanism in which a highly trusted management layer, known as a hypervisor, runs in a privileged mode of a general purpose processor [8]. The hypervisor separates multiple independent software platforms running on top of it using the Memory Management Unit (MMU), placing each inside a virtual machine controlled by the hypervisor software. There are commercial vendors such as VMware that provide such solutions. However, traditional OS vendors such as Microsoft contain hypervisor support by default with the release of windows 8 [10].

Table 3: Advantages and disadvantages of security through software virtualization

Advantages	Disadvantages
<ul style="list-style-type: none"> ✓ Any processor with an MMU can be used to implement a virtualization solution, and some of the common rich operating systems have been ported to run on top of them. ✓ There is also no requirement for additional hardware to implement a hypervisor. Security sensitive applications can be ported to run in a secure environment running on top of the hypervisor, but outside of the view of the rich operating environment. 	<ul style="list-style-type: none"> ✓ The isolation provided by virtualization technology is restricted to the processor implementing the hypervisor. Any other bus masters in the system, such as DMA engines and Graphics Processing Units (GPUs), can bypass the protections provided by the hypervisor and thus must also be managed by the hypervisor to enforce the required security policy. This is difficult to achieve without damaging the performance of the system [8]. ✓ Virtualization ignores the security issues associated with hardware attacks, such as threats that use the debug or test infrastructure.

2.4.4 Software Access Control

Software firewall refers to techniques of using a set of rules that enforce security control based on policies that confine user programs, or processes, to the minimum amount of privileges and access to resources that they require for their execution. SELinux or Security Enhanced Linux is a typical example of such “access control” mechanism embedded in Linux distributions with security awareness. Fiorin et al. describes how SELinux is used in securing desktop operating systems in [4]. Furthermore, the authors propose hardware

architecture for enhancing security and accelerating retrieval and applications of SELinux policies in embedded processors. The proposed system uses an external hardware accelerator that aims at speeding up the most time consuming operation in policy enforcement: the lookup of rules in the policy. The hardware modules include a caching mechanism, dedicated security servers. The following figure outlines the advantages and disadvantages of using software access control based approach.

Table 4: Advantages and disadvantages of software access control

Advantages	Disadvantages
<ul style="list-style-type: none"> ✓ Implementation is hardware independent, and solely developed in software 	<ul style="list-style-type: none"> ✓ Vulnerable to many types of attacks such as DoS, and tampering. ✓ Severely affects system performance, as rules are checked prior to any kernel or I/O operation ✓ Size of the code increases significantly, and typically not suited for embedded system design and development ✓ Disadvantages of using dedicated hardware also applies

Chapter 3

Cryptographic Theory

3.1 Cryptographic Tools

Cryptography is the art and science of encryption [20]. The application of cryptography in computer security is critical and fundamental topic. The field of cryptography is vast and contains many complex algorithms and techniques. For the purposes of this paper, we focus on the simple components such as Symmetric Ciphers, Asymmetric Ciphers, Hashing, and Digital signatures to design a security system. It is important to note that the secret of a cryptographic solution is not the algorithm itself but the associated keys.

3.1.1 Symmetric Ciphers

Symmetric ciphers require the sender to use a secret key to encrypt data using a predefined cryptographic algorithm (the data being encrypted is often referred to as plaintext) and transmit the encrypted data (usually called the ciphertext) to the receiver. On receiving the ciphertext, the receiver then uses the same secret key and cryptographic algorithm to decrypt it and regenerate the plaintext. The following figure shows the process of using symmetric encryption:

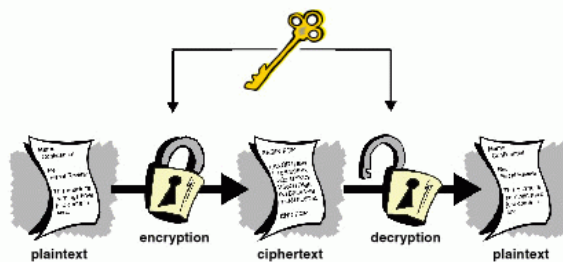


Figure 4: Symmetric Encryption

Examples of symmetric ciphers: DES, 3DES, AES, and RC4. Symmetric ciphers provide data confidentiality.

3.1.2 Asymmetric Algorithm

Asymmetric algorithms (also called public-key algorithms), use a pair of keys: one key to encrypt the data while the other to decrypt it. Encryption of a message intended for a given recipient requires only the public key to be known to the world, but decryption is only possible with the recipient's private key, which the recipient should keep secret. The following figure shows how public-key algorithm is used:

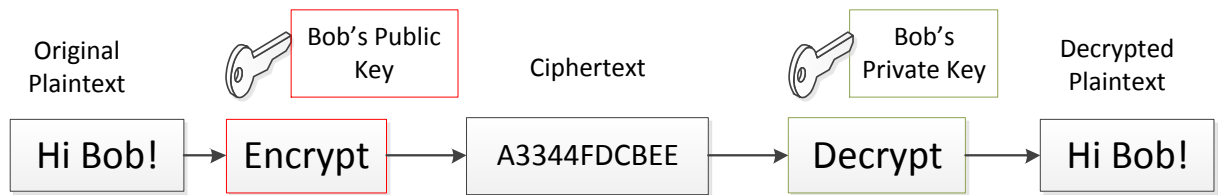


Figure 5: Asymmetric Encryption

In the figure above, if Bob is the intended recipient of the message, then the plaintext data “Hi Bob” is encrypted with Bob’s Public Key, which is advertised by Bob and then the encrypted data can only be decrypted by Bob’s private key. One popular example of asymmetric cryptography is RSA. This key-pair is also utilized in authorizing source of messages by constructing a hash of a message and then ‘digitally signing’ the message hash with its private key. The receiver the decrypts the hash with the public key, and checks the message ensuring that the sender is authorized. Digital signatures are described in detail later in this section.

3.1.3 Cryptographic Hash

A cryptographic hash is an algorithm that takes an entire message and, through a process of shuffling, manipulating, and processing the bytes using logical operations, generates a small *fingerprint* or *message digest* of the data [2]. Some examples of such algorithm are MD2, MD5 and SHA. Figure 3 below demonstrates the result of such hash function.

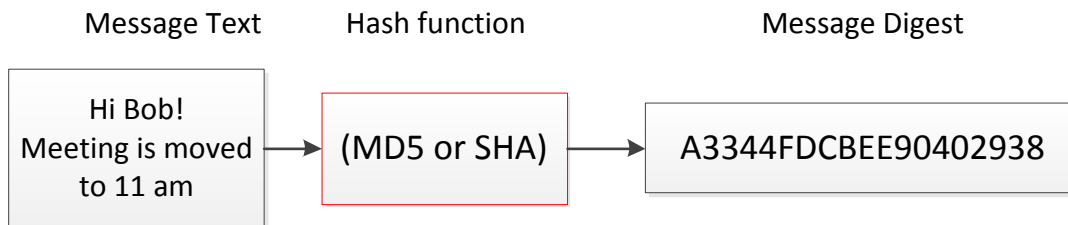


Figure 6: Cryptographic Hash

One critical feature of any hash function is that any minor change to the original message text must produce an entirely different message digest, so that any tampering can be detected. Thus hash functions are used in providing tamper proofing and/or data integrity of instructions, or communication messages or code/data in memory. Figure 4 below shows, how the end message digest changes with minor changes to the original data.

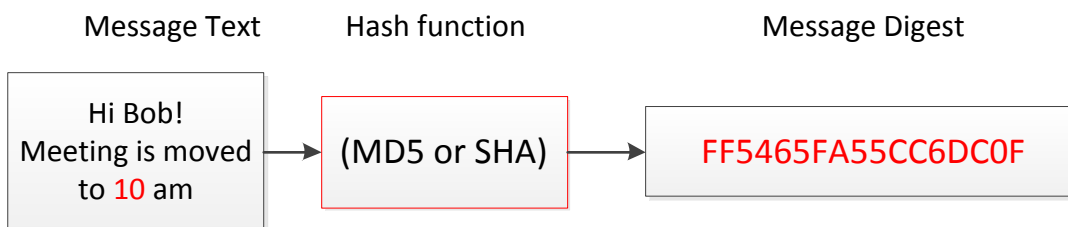


Figure 7: Tampered message Hash

3.1.4 Digital Signatures

Digital signatures, which use asymmetric cryptography and hashing functions, are implemented to authenticate the sender. This mechanism is typically used along with the encrypted data. The following figures illustrate how digital signatures are typically used. Figure 5a shows how data is signed. First hash of the data file is calculated using hashing functions (described in the previous section) such as SHA1 or MD5. Then this hash is encrypted with the private key of the sender. This encrypted file is then sent along with the data. Upon receiving the data, as depicted in figure 5b, the user decrypts the hash value with the public key of the sender, and also decrypts the data using a pre-established key-exchange mechanism. The receiver then calculates the hash of the decrypted data file, and compares to the hash decrypted with the sender public key. If the two hashes match, then the data is authenticated.

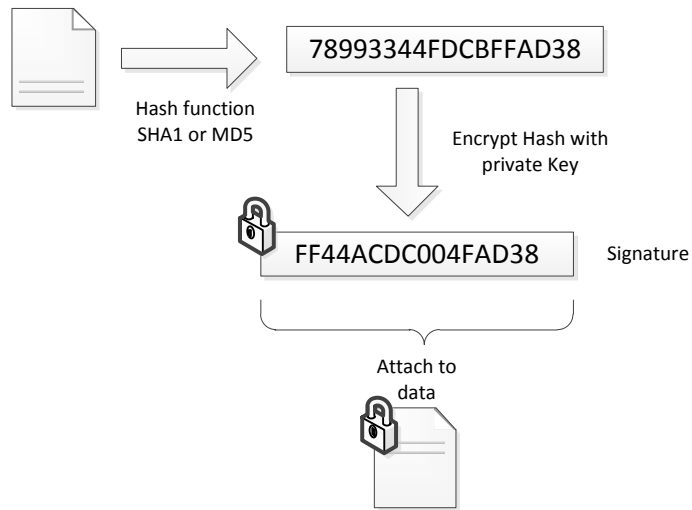


Figure 8a: Digital Signature: Signing data

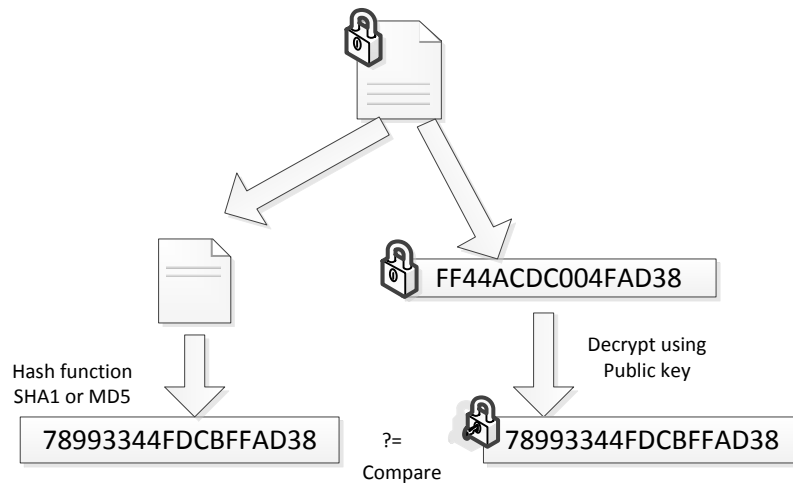


Figure 8b: Verifying signatures

Chapter 4

TrustZone Architecture

4.1 Trustzone Hardware Architecture

The more recent designs of the ARM processor core such as the Cortex A-15 core introduces concept of two virtual processors on the same core called "secure world" and "non-secure world" or "normal world". There is also a "secure monitor mode" used for switching between the two modes. These cores are called ARM TrustZone cores. On a typical ARM TrustZone core, secure world and non-secure modes coexist. Thus, Instead of protecting assets in a dedicated hardware block, the TrustZone architecture enables any part of the system to be made secure, enabling an end-to-end security solution [8]. Advantages of TrustZone are described after the hardware and software architecture is described.

4.1.1 System and bus structure

As mentioned above, in TrustZone architecture, all of the SoC's hardware and software resources are partitioned so that they exist in one of two worlds - the secure world for the security subsystem, and the Normal world for everything else. There is also a third mode called the monitor mode which switches between the two worlds. Switching is explained in section 4.1.3.

The third generation of Advanced Microcontroller Bus Architecture (AMBA) or AMBA3 AXI bus fabric contains logic that ensures that no secure world resources can be accessed by the normal world components, enabling a strong security protection barrier to be built between the two. The AMBA3 AXI bus implements two new signals - ARPROT[1] and AWPROT[1] [8]. These indicate whether the current read or write transaction is secure or non-secure (LOW: Secure, and HIGH: Non-secure). Figure 9 below demonstrates typical system architecture with isolated secure blocks and as well as shared blocks.

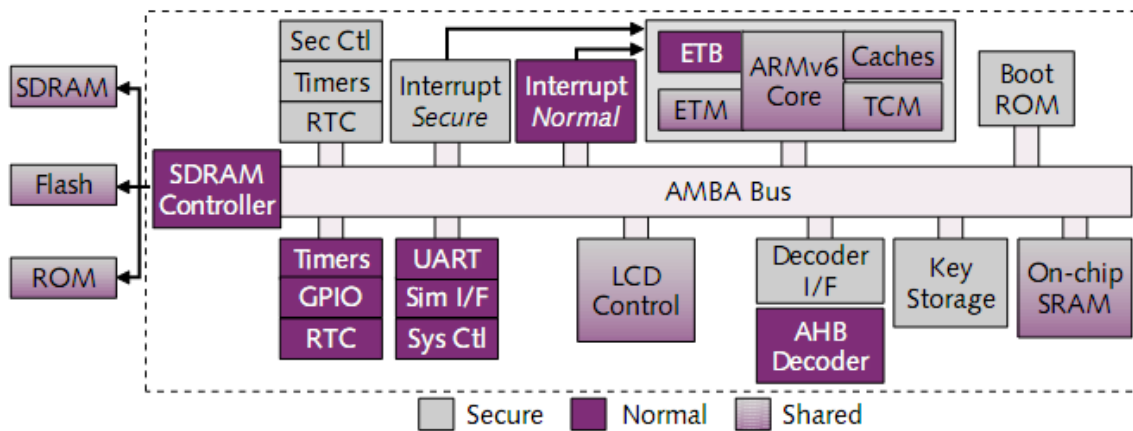


Figure 9: Typical system block diagram with share resources

In the figure above, key storage, boot ROM and decoder blocks are executed only under the secure world and normal world applications cannot access these resources. However, the cache and SRAM are shared between the two. Memory component sharing will be explained in details in section 4.1.4.

4.1.2 Register Space
















Typically arm cores contain 16 registers including stack pointer, link register, program counter and 13 general purpose registers per user mode as outlined in table 5. There are 7 user modes supported.

Table 5: Arm Processor Modes






Processor Mode	Description	Privileged
User	User applications	No
Supervisor	Kernel	Yes
System	Special version of user mode - RW access to status register	Yes
Abort	Memory access violation	Yes
Undefined	Undefined instruction	Yes
Fast interrupt request (FIQ)	Higher interrupt level	Yes

Interrupt request (IRQ)	Lower interrupt level	Yes
--------------------------------	-----------------------	-----

There is also CPSR/SPSR status registers and system control registers (CP15) for the privileged modes. Figure 10 below shows the register specification for six modes of operation in arm cores.

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8_fiq	R8	R8	R8	R8
R9	 R9_fiq	R9	R9	R9	R9
R10	 R10_fiq	R10	R10	R10	R10
R11	 R11_fiq	R11	R11	R11	R11
R12	 R12_fiq	R12	R12	R12	R12
R13	 R13_fiq	 R13_svc	 R13_abt	 R13_irq	 R13_und
R14	 R14_fiq	 R14_svc	 R14_abt	 R14_irq	 R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und


 = banked register

Figure 10: Arm register specificalton

For the TrustZone architecture, System Control Coprocessor (CP15) register and all other registers relevant exist in separate banked secure and non-secure world versions as well. All banked registers from figure 10 are shared between the two worlds. Security critical processor core status bits (interrupt flags) and System Control Coprocessor registers are either totally inaccessible to non-secure world or access permissions are strictly under the control of secure world. The CP15 register contains a NS bit as shown in figure 11 below.

This NS bit is set to 1 when the processor is operating in non-secure world and 0 when it is in secure world. Access to this bit is only allowed from the secured mode, and monitor mode which is responsible for switching worlds. Monitor mode is explained next.

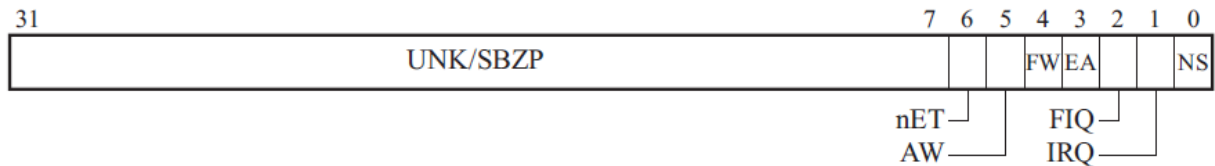


Figure 11: CP15 register contains the NS bit set according to mode of operation

4.1.3 World Switching

For the purpose of interfacing between secure and non-secure world a special mode called “Monitor Mode” is introduced. Monitor mode manages transition between the two modes and also manages the NS (Not Secure) bit in Secure Configuration Register in CP15. Figure 12 below shows all three modes.

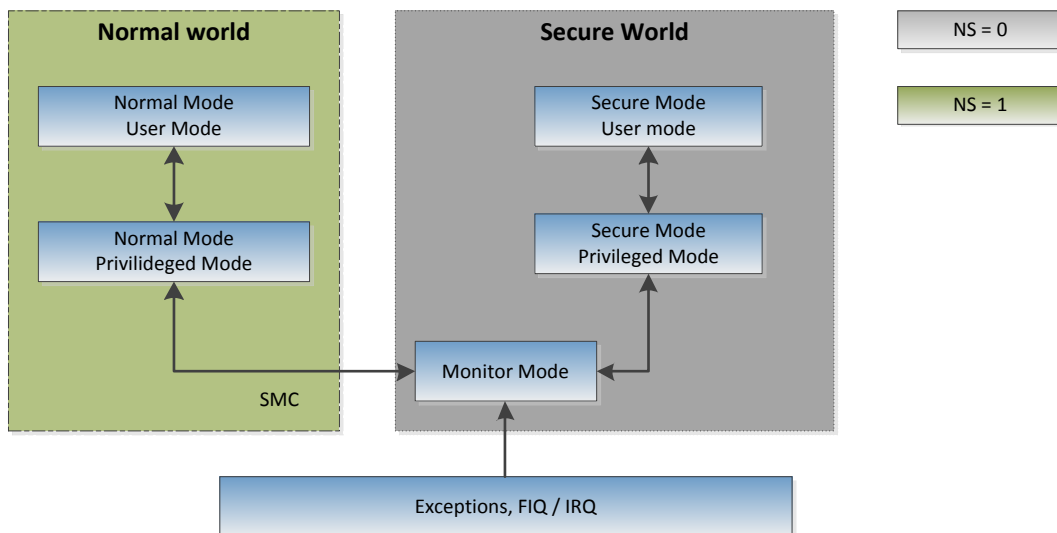


Figure 12: Secure world, normal world and monitor mode

ARM introduced a special instruction is called SMC, Secure Monitor Call, which switches execution from one world to the other. Normal mode software running in the non-secured World can execute the SMC instruction to invoke a switch to secure world through the

monitor mode software. The switch can also be configured to be triggered by IRQ, FIQ, external Data Abort, and external Prefetch Abort exceptions.

In most designs its functionality will be similar to a traditional operating system context switch, ensuring that state of the world that the processor is leaving is safely saved, and the state of the world the processor is switching to is correctly restored. Typically all general purpose ARM register, any coprocessor register and any world-dependant processor configuration state in CP15 must be saved and restored during context switching.

Normal world and the secure world execute in a time-sliced fashion. This removes the need for a dedicated security processor core, which saves silicon area and power, and allows high performance security software to run alongside the normal world operating environment.

4.1.4 Memory and Cache

The major component of the L1 memory system is the Memory Management Unit (MMU), which maps the virtual address space that is seen by the software to the physical address space [12]. The address translation is managed using a software-controlled translation table, which details which virtual address corresponds to each physical address, and some other attributes about the memory access, such as access permissions. There are two virtual MMUs on the arm TrustZone architecture. Thus each world maintains a local set of translation tables and virtual address and physical address mappings.

ARM processors with TrustZone tag entries in the Translation Lookaside Buffers (TLBs) that cache the results of translation table look up. This allows for non-secure and secure worlds to co-exist in the TLBs and aids in fast context switching between worlds.

It is equally important to cache data and instructions for both the worlds to allow for fast context switching as cache does not have to be flushed and restored during the switch. To enable this, the L1 and L2 (if applicable) processor caches have the NS tag bit which records

the security state of the transaction that accessed the memory. However, both worlds are allowed to clear cache lines upon memory shortage. Also, this allows for data to be shared between non-secure and secure worlds. Figure 13 below shows the how cache and TLBs are tagged.

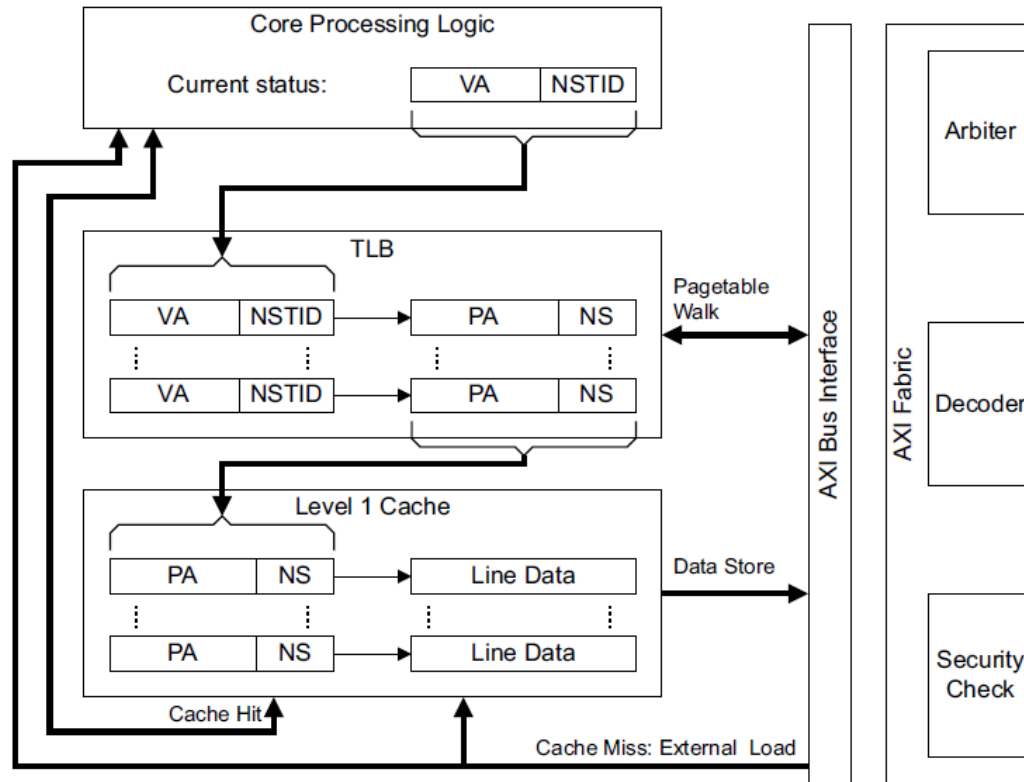


Figure 13: Cache and TLBs

Figure 14 below shows memory organization with an example application that decrypts data which is passed in as encrypted from the Normal World. The encrypted data is put in a common non-secure memory block which is mapped to the same physical address space in both normal and secure worlds. The data is fetched in the secured world, and then decrypted using secured services and processed according to the application type such as DRM, Banking etc.

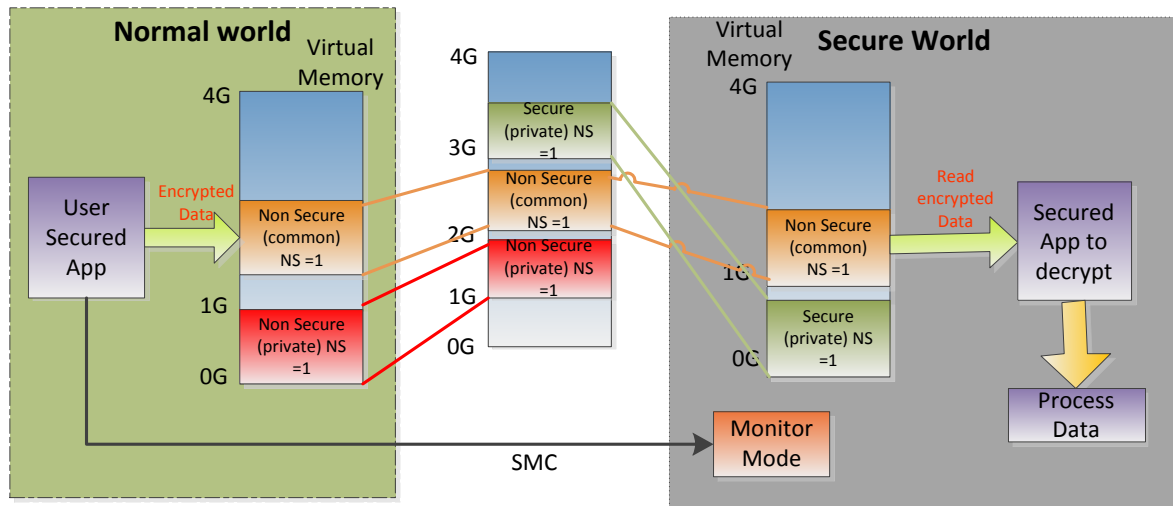


Figure 14: TrustZone memory access example

4.1.5 Interrupts

ARM processor with TrustZone extension contains an Interrupt Controller (GIC) that provides secure interrupt sources which cannot be manipulated by the normal world software. All interrupts are trapped in the monitor mode. If the core is in the other world and an interrupt occurs, the monitor software causes a context switch and jumps to the restored world, at which point the interrupt is handled as shown in figure 15. Also as an added measure, a configuration register in CP15 to prevent any Normal world software modifying the F (FIQ mask) and A (external abort mask) bits in the CPSR and therefore blocking the apps from masking out secure world interrupts.

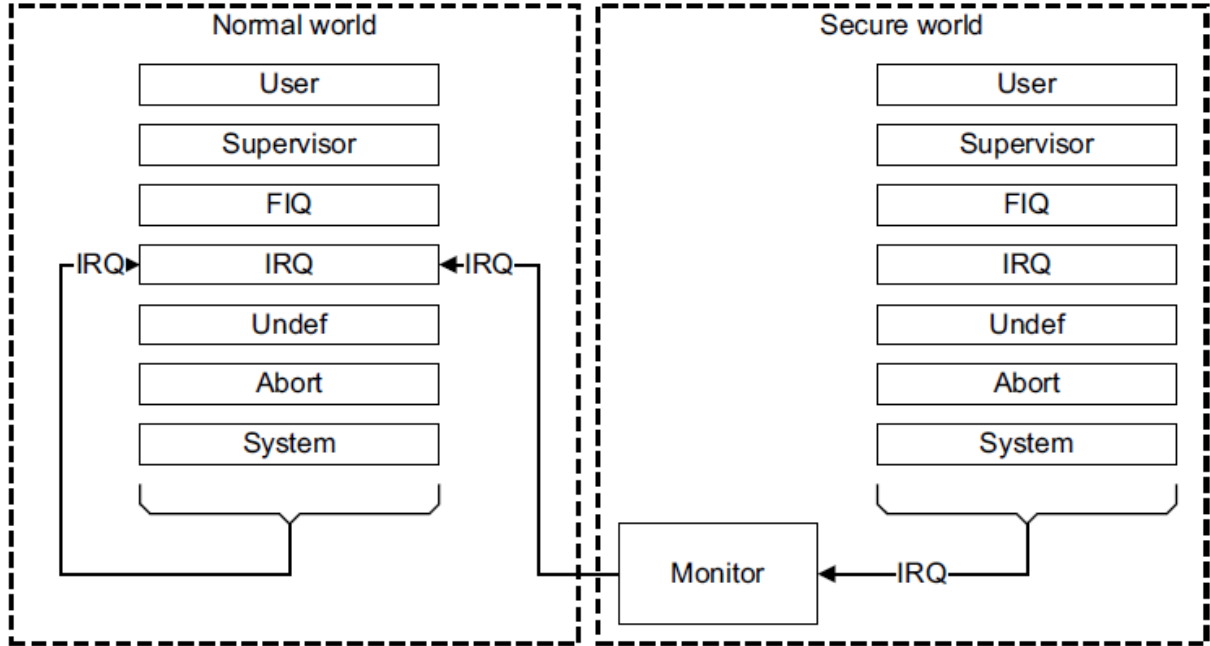


Figure 15: Interrupt propagation between worlds

4.1.6 Peripherals

One of the most useful features of the TrustZone architecture is the ability to secure peripherals, such as interrupt controllers, timers, and user I/O devices. Thus it is possible to secure entire system instead of providing a secure data processing only. A secure interrupt controller and timer allows a non-interruptible secure task to monitor the system, a secure clock source enables robust DRM, and a securable keyboard peripheral enables secure entry of a user password [8].

TrustZone architecture includes a peripheral bus known as the Advanced Peripheral Bus (APB), which is attached to the system bus using an AXI-to-APB bridge as shown in figure 16 below. The AXI-to-APB bridge hardware is responsible for managing the security of the APB peripherals by rejecting transactions of inappropriate security setting and must not forward these requests to the peripherals.

The bridge contains address decode logic that generates the APB peripheral select based on the incoming AXI transaction. The bridge includes a single TZPCDECPROT input signal for each peripheral that is located on the bus. This signal is used to determine if the peripheral is configured as secure or non-secure. These bridge input signals can be tied persistently at synthesis time or can be dynamically controlled via a trusted peripheral, such as the TrustZone Protection Controller (TZPC), to allow dynamic switching of security state at run-time.

Figure 16 demonstrates how 4 peripherals are controlled. The TZPC is configured as always Secure [DESCPROT = 0], the Timers and Real-Time Clock (RTC) as always Non-secure [DESCPROT = 1], and the Keyboard and Mouse Interface (KMI) have a programmable security state under software control. Secure world software can program the TZPC at run-time to change the signal input to the AXI-to-APB Bridge to switch the KMI from Secure to Non-secure or vice versa. This allows the system to capture password entered by user securely.

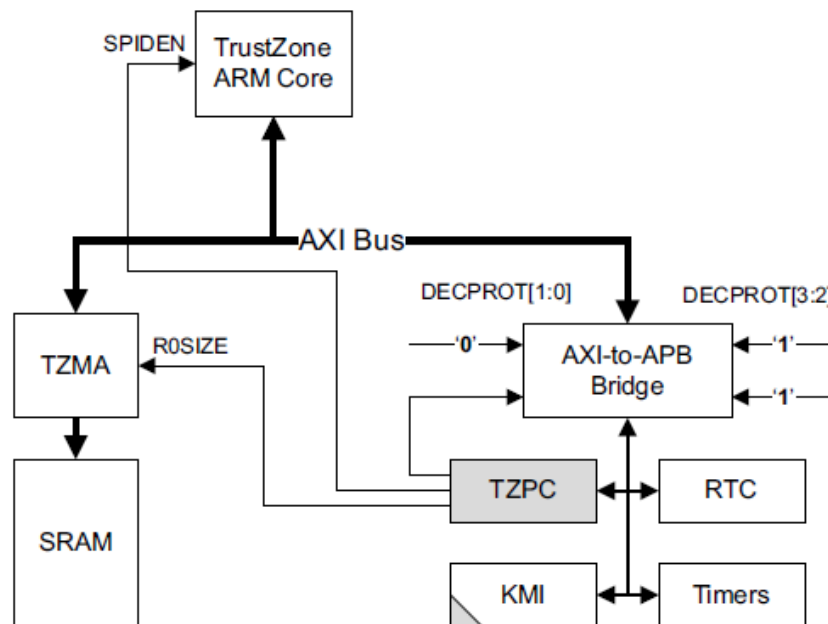


Figure 16: Trustzone and secured peripherals

4.2 Trustzone Software Architecture

Figure 17 below shows the software architecture of trustzone. The normal world and secure world kernels operate in stand alone mode. The secure world is responsible for secure booting of the normal world kernel. Secure booting is out of scope of this paper, but Kai et al describes secure booting of embedded systems in [13] .

Typically a normal world user app would make a call to secure world for secured services such as cryptographic functionalities. The user app interfaces with the TrustZone driver, which makes a call to the monitor mode software using the SMC instruction described earlier. The normal world kernel or the user application is responsible for copying the data to be processed into a shared memory region described in section 4.1.4. Upon finishing secure operation, the return value is returned to the user app. Since both the OS operate in standalone mode, multi-threading is possible in each of the worlds. Therefore, the choice of OS is flexible, for example the normal world OS can be running Windows with Linux running in the secure world. The TrustZone Driver is therefore modified based on the configuration and the secure world implementation does not need to change.

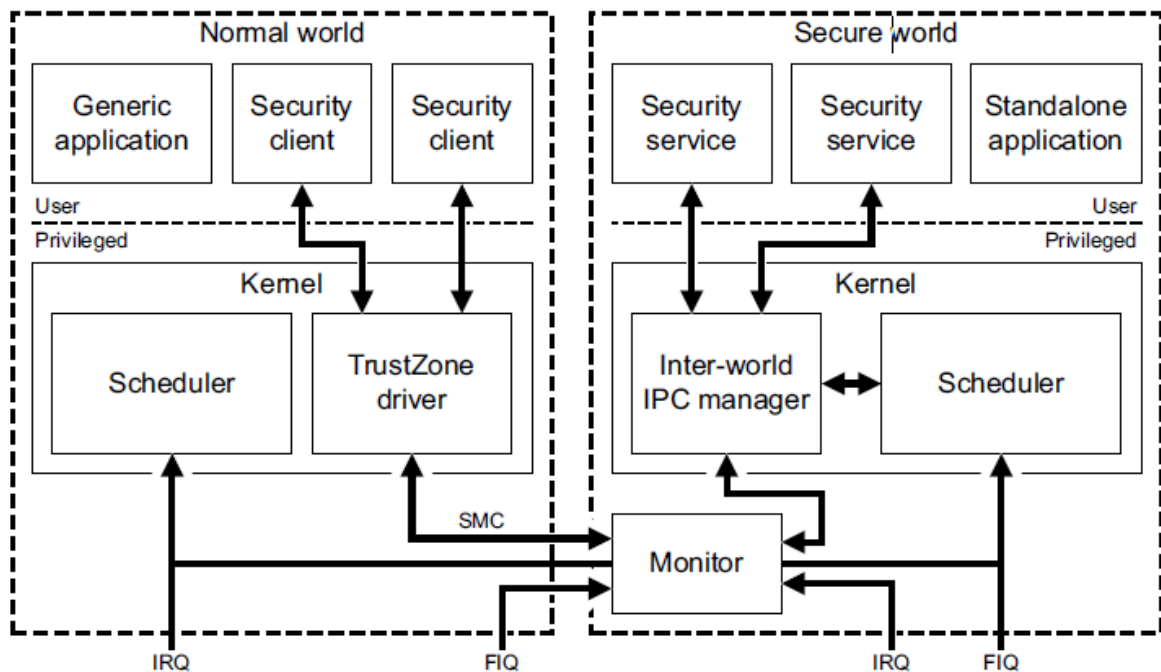


Figure 17: Trustzone Software Architecture

4.3 Benefits of Trustzone

The main benefits as outlined in [15] of TrustZone are:

1. TrustZone provides secure storage for keys, and data.
2. Full bus-bandwidth access is provided to all storage areas to provide fast memory access speeds, which improves overall system performance.
3. Since TrustZone solution consists of software and hardware elements, it provides flexibility to allow customizations upgrades to the secure system even after the SoC is finalised.
4. Any peripheral channel such as, the user interface, SIM and smart cards as well as audio output can be secured. For the non-secure world, TrustZone can enable security through integrity checking for all the features within a SoC device. For example, decoded DRM audio can be protected as it is passed to non-secure audio drivers by integrity checking the relevant part of the OS infrastructure.

Chapter 5

Example Application: Mobile Movie Ticket Application

This section explains design of a secure system based on the TrustZone architecture described above.

With the evolution of the smart phones, tablets, and e-readers mobile commerce has become a prominent industry. Nokia defines mobile commerce as the use of mobile handheld devices to conduct any electronic transaction or information interaction that leads to transfer of value in exchange for information, services or goods over wireless networks in a released whitepaper [5]. Examples of m-commerce application include mobile financial services, mobile context services, on-line games, tickets and mobile shopping. A lot of research has been done to ensure secure communication over the air and through network backbone. However, according to Group Inc. it is predicted that 85 percent of wireless security incidents will be device related rather than over-the-air related [6]. Therefore content security can be considered as the most important area of handheld security. This is where the advantage of TrustZone based security solutions is crucial. Hussin, Coulton and Edwards outlined the design for a secure mobile commerce application in [7]. Even though the design presented in this paper follows the same set of requirements and application, the methodology and security engineering practices is different. The design attempts to take advantage of the concept of two worlds in one processor. In the next few sections, first system requirements are outlined followed by software flow which is based on top of the TrustZone kernel ported. The hardware setup for simulation is described in the next section.

5.1 System Requirements

Let's consider a system to purchase movie tickets. The user browses and selects a movie a particular theatre for a specific date and time. The user would then pay for the ticket using mobile banking which is assumed to be secured for the purposes of this example. Upon successful completion of the transaction the user needs to receive an electronic receipt as proof of purchase, which is referred to as the "mobile ticket". The mobile ticket must possess the following characteristics:

1. The ticket received cannot be falsified
2. The ticket can only be valid for a specific date and time
3. The ticket cannot be used twice and has to be unique
4. The ticket must be verified to be issued by the authorized movie operator only

To ensure such security requirements, one design based on TrustZone technology is provided.

5.2 System Block Diagram

Figure 18 below shows the system block diagram for a TrustZone based movie ticket application. The operator is connected to the handheld device over a secured network. The handheld devices have a normal world application which interfaces with the operator. The normal world app is also a secure application (described in the next section). The normal world app talks to the secured world which provides security services such as cryptographic library, secure storage of keys through the predefined TrustZone API (Appendix C). There may also be attached peripherals such as a keypad, or touch screen to get user input and/or display secured content. Note that a secured protocol such as HDCP can be deployed to ensure output protection for the screen, but that is out of scope of this discussion and it is assumed that the output is secure. User input is made secured by using the peripheral security feature as explained earlier in section 4.1.6.

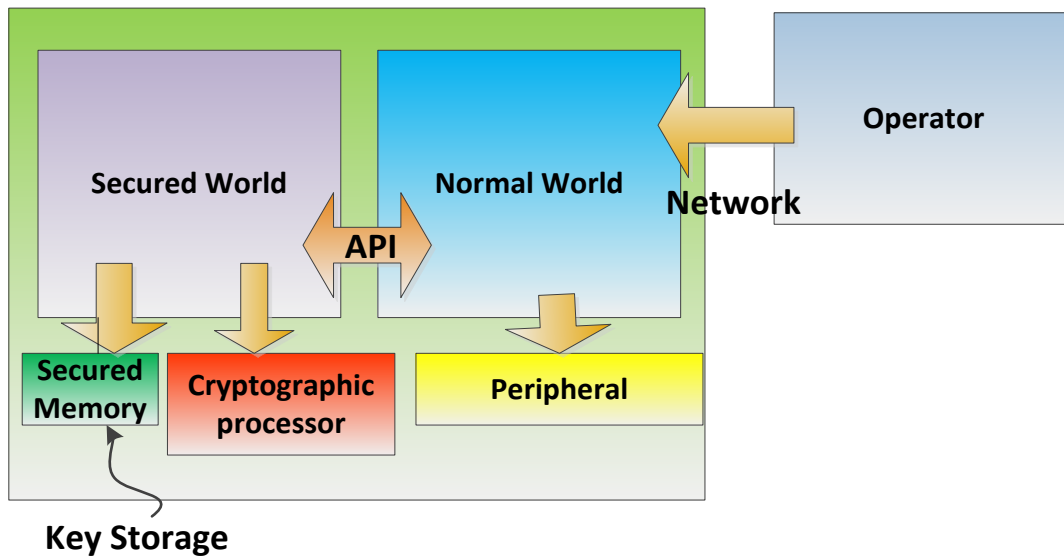


Figure 18: System Block Diagram

The cryptographic processor is responsible for handling all cryptographic computations and algorithms such as AES, RSA, SHA-1 etc. OpenSSL implementation is used to simulation this behavior but with real hardware crypto processing is accelerated.

5.3 Operating System and Cryptographic Processor

Linux Kernel is ported to the ARM Cortex A15 processor with ARM TrustZone support. The scheduler is modified to schedule secure world and normal world applications in a time sliced manner [21]. The normal world communicates with the secure world through the TrustZone API and OpenTZ driver. The cryptographic processing is done with the software implementation of OpenSSL [22]. When actual hardware is used the a separate cryptographic accelerator can used which can be called through OpenSSL directly without code change.

5.4 Normal World User Application

The normal world user application interfaces with the network operator over a secured network connection. The user app also connects to a third party payment service such as Paypal or Interac. Since this secured app is place in system memory it must provide the

security requirements outlined in section 2 such as tamper resistance, data confidentiality, integrity, and authenticity.

5.4.1 Tamper Resistance

Tamper resistance is provided in two phases: 1) load-time integrity and 2) run-time integrity. Load time integrity mechanism is rather simple. A checksum is pre-calculated during compile time and is embedded in the program itself. The execution binary is processed by another tool written to calculate the CRC and insert it in the execution header. The program then calculates the checksum upon start up and continues execution only the calculated checksum matches the embedded checksum. Since this is a crucial part of the integrity check, the approach to ensure this process is secured needed to be sophisticated. For example, if there is code such as the following figure then it is very easy break the code.

```
if (crc_caluclated != crc_stored){  
    break;  
}
```

Figure 19: CRC comparison code

The code snippet in figure 19 looks like the figure 20 in assembly. An attacker simply has to change the je instruction (opcode 0x74) at offset 8048445 to a jmp instruction (opcode 0xEB) to defeat this protection.

```
804842b: ff 75 fc pushl -4(%ebp)  
804842e: ff 75 f8 pushl -8(%ebp)  
8048431: e8 f2 fe ff ff call 8048328 <crc> ;call crc32( )  
8048436: 83 c4 10 add $0x10,%esp  
8048439: 89 45 f4 mov %eax,-12(%ebp)  
804843c: 8b 45 f4 mov -12(%ebp),%eax  
804843f: 3b 05 d0 83 04 08 cmp 0x80483d0,%eax ;compare result  
8048445: 74 22 je 8048469 ;jump-if-equal
```

Figure 20: CRC check instruction level

To counter this type of attack, security through obfuscation is chosen, where a table of function pointers is utilized to hide the actual value of the checksum. For example, if the calculated checksum is 0x09A1DE9F then in the table the following entries would contain actual valid data byte1[0], byte2[9], byte3[10], byte4[1], byte5[13], byte6[14], byte7[9], byte8[15]. Each of these contains a pointer to the function that will process the next nibble in the checksum, except for b8[15], which contains a pointer to the function that is called when the checksum has proven valid. This theoretical concept is shown in figure 21 below:

```
byte1[16] = { crc_nib2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
byte2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, crc_nib3, 0, 0, 0, 0, 0, 0, 0 },
byte3[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, crc_nib4, 0, 0, 0, 0, 0 },
byte4[16] = { crc_nib5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
byte5[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, crc_nib6, 0, 0, 0 },
byte6[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, crc_nib7, 0, 0 },
byte7[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, crc_nib8 },
byte8[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, crc_good };
```

Figure 21: Check sum function pointer table

However, in reality, the 0's in the function table also poses a threat to breakdown this function pointer table. Thus, a lot of dummy functions are populated at random with a random initial vector to obfuscate the code. A screenshot of the out of this example is given in figure 22.

```

$ ./mainProg.exe
Allocating block of 1024 (allocation 0)
Allocating block of 1024 (allocation 1)
Allocating block of 1024 (allocation 2)
Allocating block of 1024 (allocation 3)
Allocating block of 1024 (allocation 4)
Allocating block of 1024 (allocation 5)
Allocating block of 1024 (allocation 6)
Allocating block of 1024 (allocation 7)
Allocating block of 1024 (allocation 8)
Allocating block of 1024 (allocation 9)
CRC Table function pointers
|0x004060D6|0x0040620E|0x004060D6|0x004061A6|0x0040613E|0x00406006|0x00406006|0x0040620E|0x0040620E|0x004060D6|0x004060D6|0x0040613E|0x00406006|0x0040620E|0x004060D6|0x004061A6| |
|0x00406006|0x0040613E|0x00406006|0x004061A6|0x004060D6|0x00406006|0x0040620E|0x0040620E|0x004060D6|0x00406006|0x0040613E|0x004060D6|0x00406006|0x0040620E|0x004060D6|0x004061A6|
|0x00406006|0x0040606E|0x0040613E|0x00406006|0x0040620E|0x0040613E|0x004061A6|0x0040606E|0x0040613E|0x0040606E|0x0040613E|0x0040606E|0x004061A6|0x0040606E|0x0040613E|0x0040606E|
|0x004061A6|0x0040613E|0x0040620E|0x00406006|0x004061A6|0x0040613E|0x0040613E|0x00406006|0x0040613E|0x004060D6|0x004061A6|0x0040620E|0x00406006|0x00406006|0x0040606E|0x0040606E|
|0x0040613E|0x0040613E|0x004061A6|0x0040606E|0x0040606E|0x00406006|0x0040606E|0x00406006|0x004060D6|0x0040606E|0x004061A6|0x0040613E|0x0040613E|0x0040620E|0x00406006|0x00406006|
|0x0040606E|0x0040606E|0x0040620E|0x00406006|0x0040620E|0x004061A6|0x0040620E|0x0040613E|0x0040613E|0x004060D6|0x0040606E|0x00406006|0x004061A6|0x004061A6|0x00406006|0x00406006|
|0x004060D6|0x00406006|0x00406006|0x0040606E|0x0040606E|0x0040606E|0x0040620E|0x0040620E|0x0040606E|0x0040606E|0x004060D6|0x0040613E|0x004061A6|0x004061A6|0x004060D6|0x00406006|0x00406006|
|0x0040620E|0x00406006|0x0040606E|0x0040620E|0x0040606E|0x0040620E|0x004060D6|0x0040613E|0x004060D6|0x0040620E|0x0040606E|0x004060D6|0x004060D6|0x004060D6|0x0040613E|
CRC is 09A1DE9F
CRC is valid.

```

Figure 22: Actual function pointer table for CRC 0x09A1DE9F

A separate program is written to calculate the static CRC of the user application. The static CRC is also stored in the secure memory and checked when the user application is launched.

5.4.2 Data confidentiality

Tamper resistance provides protection against changing the data or code used by the software. However, it is also important to protect data that resides in system memory. There are few techniques to do secure memory for example by using memory in external hardware such as a GPU. For key storage and other data storage purposes the memory space in the secure world is used as described in the TrustZone Hardware Architecture section (4.1.4). For added security, a secure memory library is used to protect data in the stack as infrequently used data is usually swapped to disk. This library ensures that all memory allocated by the program is locked, so that it cannot be swapped out. Code for this library is included in the Appendix.

Another level of security is added by checking for the CRC dynamically. A CRC “lattice” is calculated from the Read only sections of the binary and constantly checked during runtime for modifications. The functions implementations reside in the Read only section and thus cannot be modified by an attacker.

5.4.3 Secure World Kernel API

Normal world app requests cryptographic services through the predefined API. The services could include cryptographic calls, and data processing, but in this application, encrypted data is passed to the secure world for decryption and processing as explained in the next section. So the primary purpose of this application is to ensure secure data processing.

5.5 Key-exchange and software flow

This section describes the software flow of the mobile ticketing application and how cryptographic blocks and design techniques described in section 3. Figure 23 below shows the flow diagram. For rest of this section the following notation format is used as outlined in Table 6.

Table 6: Notations

Notation	Meaning
$K_{pub}(operator)$	1024 bit RSA public key of operator
$K_{priv}(operator)$	1024 bit RSA private key of operator
K_{AES}	128 bit AES key for encryption/decryption
Client	Normal World Application
Hash(plaintext)	SHA-1 Hash of {Plain Text}

As described earlier, the user browses for movie tickets and process secure payment through third party vendors. The scope of secure payment is out of this discussion, but more details can be found in [8]. After the operator receives a request, pre-generated public RSA keys $\{K_{pub}(operator), K_{pub}(client)\}$ are exchanged between the normal world app and the operator software. Then the operator processes the ticket request and generates a ticket in plain text that contains the current time, movie name, location, and time. The format of the file could be complex with watermark, and a bitmap image. The plain text information is then sent to the secured app along with the keys. The application software then generates a random 128 bit (16 bytes) key, K_{AES} . It is assumed that the random number generator is a True Random Number (TRN) generator. The properties and nature of TRN is out of the

scope of this project. The movie ticket, plain text format in this example, is then encrypted with K_{AES} . This encrypted ticket is referred to as “cipher ticket”.

Then the SHA-1 hash of the cipher ticket, $\text{Hash}(\text{cipher ticket})$, is calculated and signed with the private key of the client and sent along with the cipher ticket as digital certificate as described in section 3.4. Also, K_{AES} or key to encrypt the plaintext ticket is encrypted with $K_{pub}(\text{client})$. After all the above steps are completed, the cipher ticket, digital certificate, and the encrypted key are sent over the secured network channel to the operator.

On the operator side, first the encrypted AES key, K_{AES} , is decrypted with $K_{priv}(\text{operator})$. Then using the decrypted key, the cipher movie ticket is decrypted. The hash value of this decrypted ticket is calculated and stored. The digital signature received is decrypted with $K_{pub}(\text{client})$ and the received hash value is obtained. Then the calculated hash value of the decrypted ticket and the received hash value are obtained. If the two values match, then the transmission is marked as successful, and authenticated.

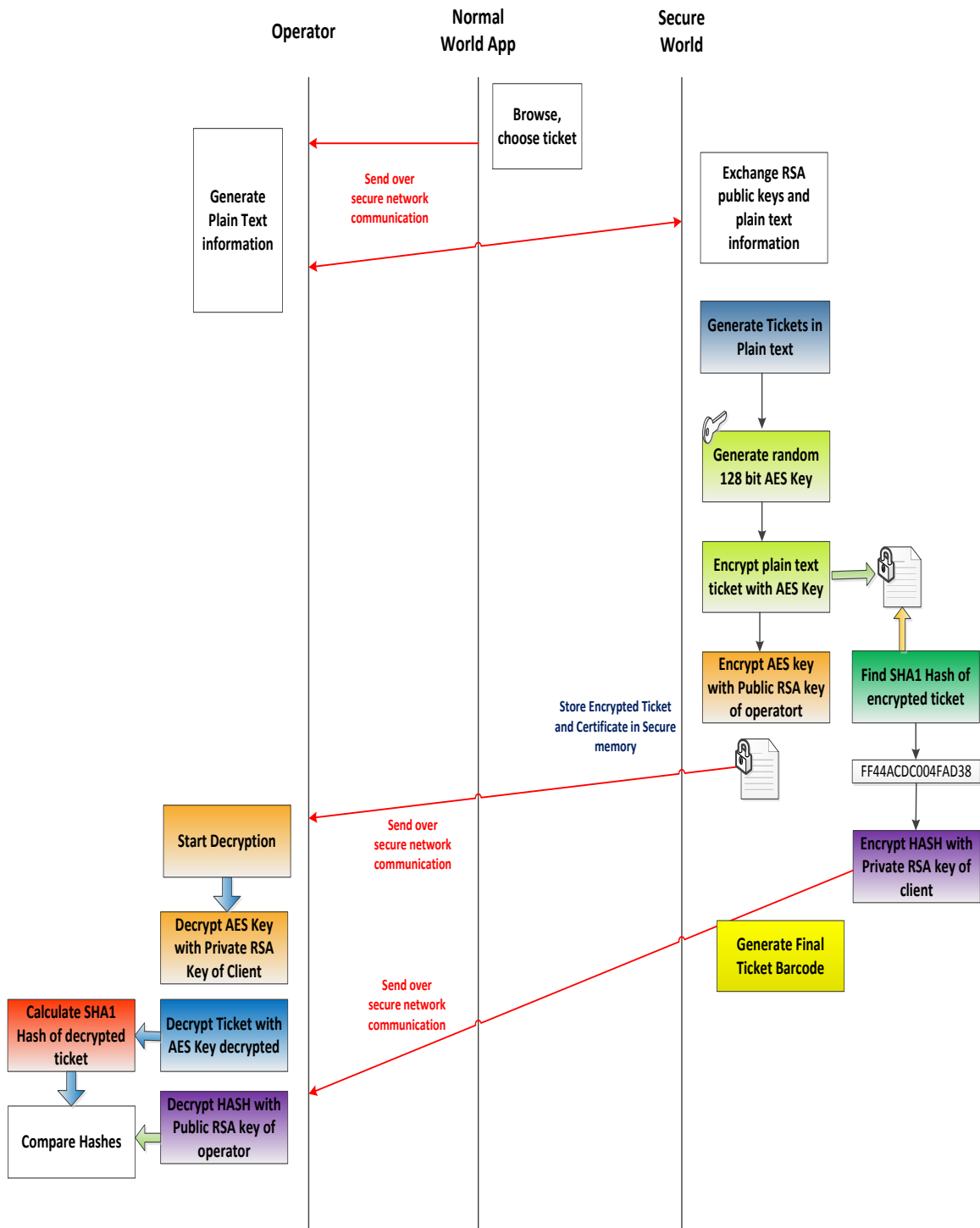


Figure 23: Key exchange and software flow

Chapter 6

Simulation Results

The concepts presented in this paper are simulated in ARM fast model simulation environment. The cost of real hardware is significantly high, thus simulation used as proof of concept. For this project, Versatile Express (VE) baseboard component is used with Cortex A15 processor with 1 core. This processor has trustzone support enabled by default. The details of the simulation environment can be found in [9]. Figure 24 below shows the System Canvas block diagram editor being used to draw the schematic representation of the VE, and figure 25 shows the schematic in detail.

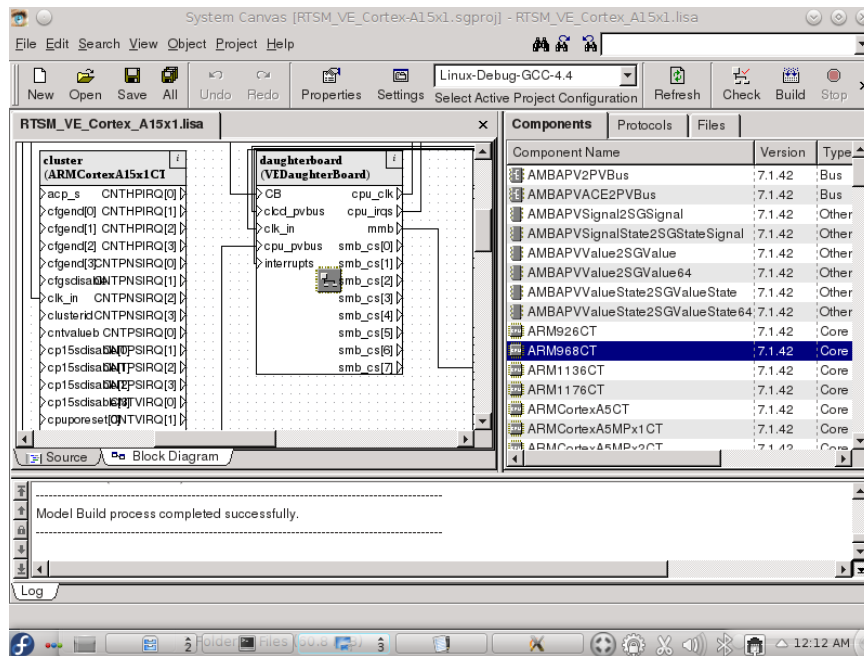


Figure 24: System Canvas Simulation

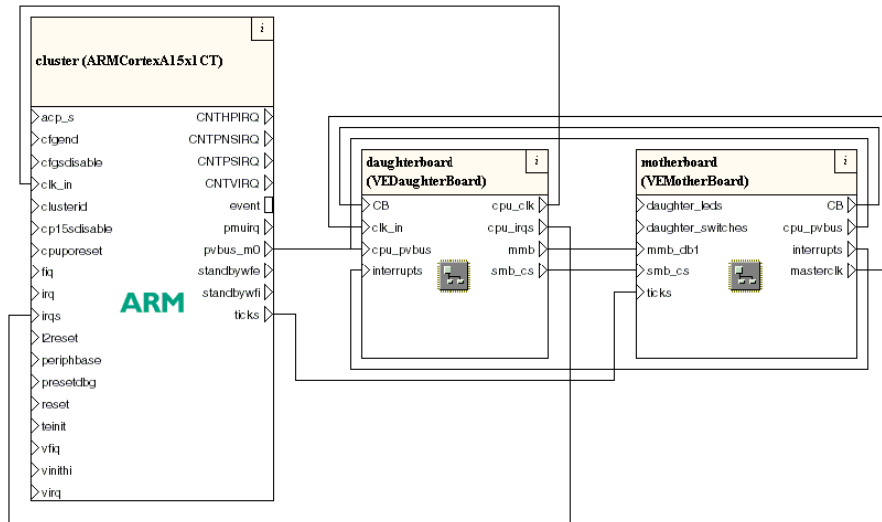


Figure 25: Schematic of VE with Corex A15 processor

Figures 26-28 show screenshots of how Linux is loaded on this platform.

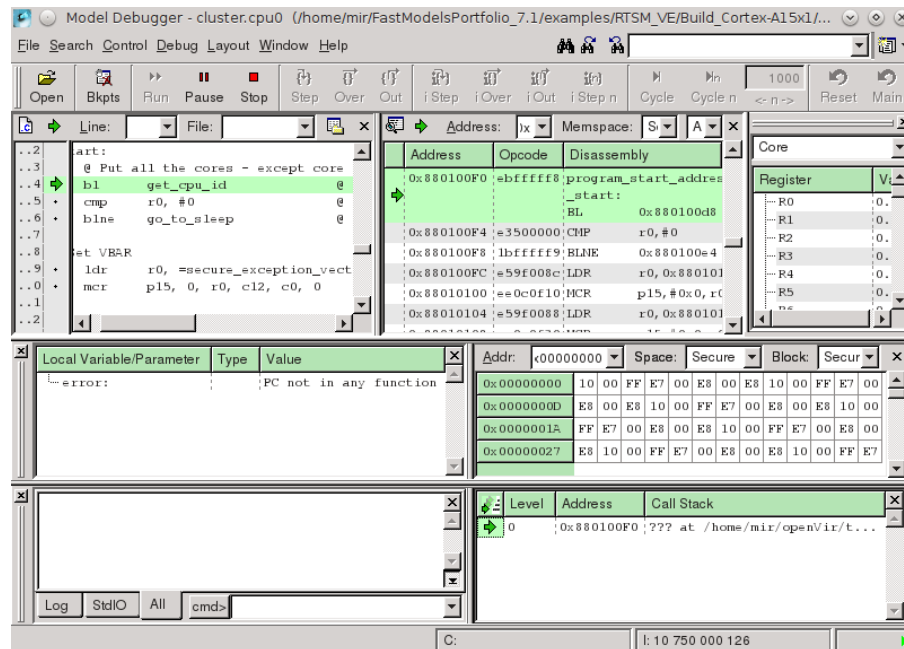


Figure 26: Memory view

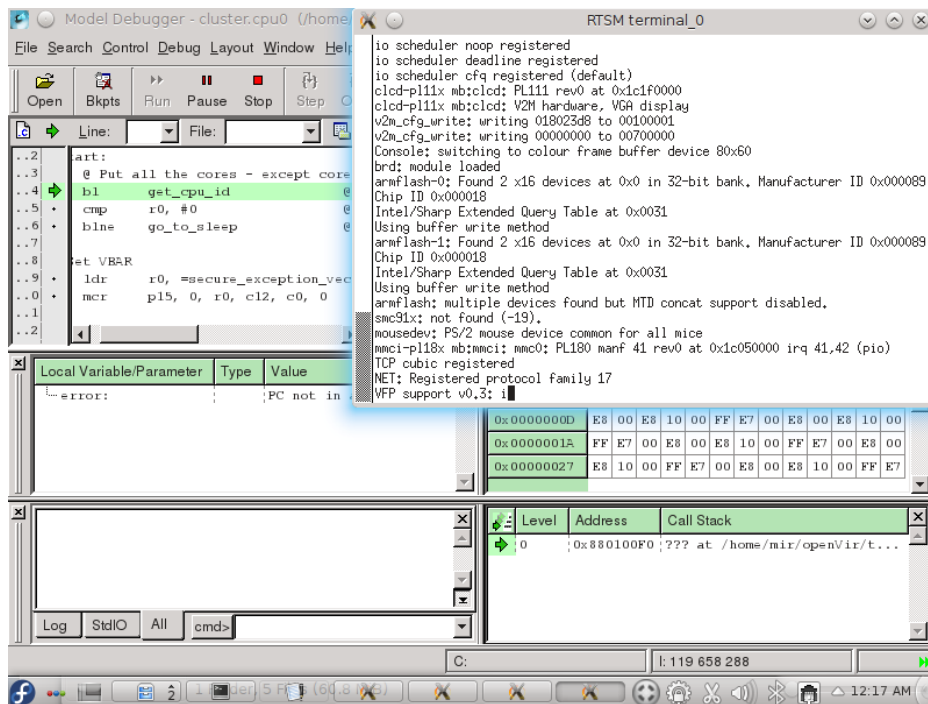


Figure 27: Linux booting

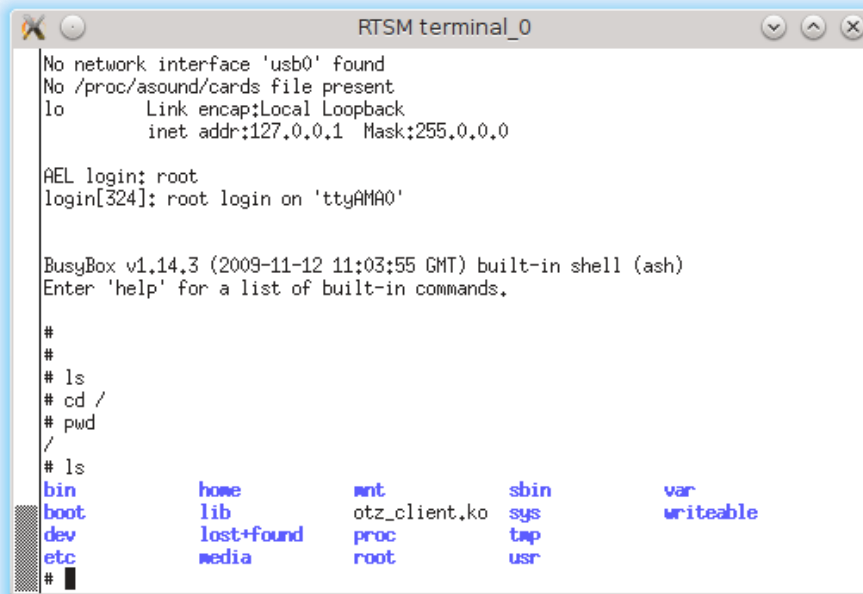


Figure 28: Linux terminal with busybox

The Linux kernel 2.6.34 patched code and application code is attached in Appendix B. When the normal world application is run and key exchange and negotiation is successful, a number of files containing the AES keys, RSA keys are dumped. The output files are attached with this report. The final output is the barcode generated from the encrypted ticket. The figure below shows the barcode generated using Zint – Grid Matrix (AIM Standard),

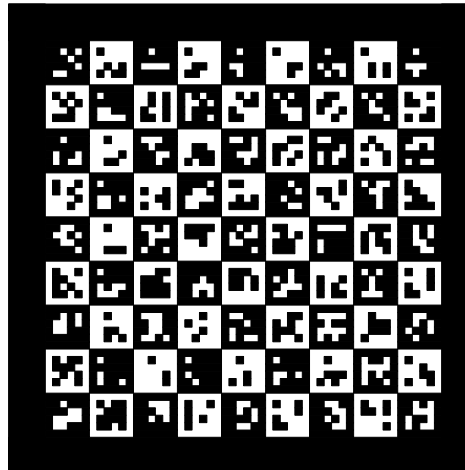


Figure: Grid Matrix generated barcode form encrypted data

6.1 Security Threat Analysis

A number of security threats were analyzed and simulated. Buffer Overflow attack, Static code tampering, dynamic code tampering, ROP (Return Oriented Programming) exploitation and Man-in-the middle attacks are carried out and the program behavior is noted.

6.1.1 Buffer Overflow

Buffer Overflow attack is the most common mechanism of attack in systems where the binary can be exploited by an attacker. The attack is explained by Cown et al in [24] as is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. A typical solution is to use the compiler to protect against such attack. All the binaries for this project were compiled with `-fstack-protector` option to enable protection from buffer overflow attacks.

6.1.2 Static Code tampering

In this mechanism, the program binary is modified to alter code data using a script and hex modification tools. A number of patterns of code/data were modified either randomly or at predefined intervals. In most cases, the executable crashed without processing any keys or secrets. If however, the modifications did not crash the program, the CRC check against the CRC value calculated and pre-injected failed and the program exit.

6.1.3 Dynamic Code tampering

If code/data were modified using a debugger while the code was running, the CRC checks would also fail upon rechecking the value and the program would terminate itself. A full proof monitoring mechanism, i.e, protection against an attack where the monitoring mechanism itself is compromised is not implemented yet.

6.1.4 Return Oriented Programming (ROP) Exploitation

Another popular attack on binaries is the Return Oriented Programming (ROP) exploitation where instead of jumping to the beginning of a library function, the attacker chains together existing sequences of instructions (called Gadgets) that have been previously identified inside existing code [23]. ROP analysis on the binary itself exposed the following Gadgets,

```

..... => mov %eax,%e?x)

- 0x08048be0(null) => pop %eax ; ret

- 0x08048c1b(null) => pop %ebx ; ret

- ..... => pop %ecx

- ..... => pop %edx

- ..... => xor %eax,%eax

- 0x08048bb0(null) => inc %eax ; ret

- ..... => inc %ax

- ..... => inc %al

- ..... => int $0x80

- ..... => sysenter

- 0x08048fd3(null) => pop %ebp ; ret

- 0x0804c380(null) => .data Addr

```

As a result, protection against ROP exploitation is needed. Onarlioglu et al, describes a compiler-based approach that eliminates all unaligned free-branch instructions inside a binary executable, and protects the aligned free-branch instructions to prevent them from being misused by an attacker. To militate against ROP attack, the compiler modifications could be implemented.

6.1.5 Man in the Middle Attack

For this scenario it was assumed that the attacker controls the interface between the user mode and the kernel mode components and replays the traffic requesting cryptographic services from secure mode. However, since the normal mode application is signed and certified prior to execution the validation method implemented in the kernel mode driver did not execute commands from and unauthorized source. One possible way the attacker could still get control if the private key of the signing tool is compromised. As a result, this key is to be refreshed periodically.

Chapter 7

Conclusion

Embedded system security is identified to be one of the most important parts of the design flow as more and more systems are exposed to attackers and the consequences of any breach can result in costly damages. Data confidentiality, Data integrity, Authentication, Content Security, Secure Storage and Tamper Resistance are some of the features that any secured system must provide. A few different security solutions exist today such using a dedicated hardware block, or software virtualization techniques. However, the benefits of such solutions do not always apply to embedded systems where resources such as memory and computing power are limited. TrustZone technology uses a concept of running a normal processor and secured processor (which are isolated from each other) on the same SoC to provide security services. TrustZone based solutions save silicon space but provide all necessary security features such as secured storage, secure execution, secure peripherals in performance efficient manner with robust hardware and software design explained in this paper. Also, secure boot features can also be incorporated in TrustZone without the cost of having to implement extra logic or hardware block. The upgrade process is also simplified. To validate the use of TrustZone, a mobile ticketing system design is implemented using the proposed architecture. Simulation results show that such system is an effective solution for embedded systems with memory and resource constraints without compromising the security requirements. A number of security attacks such as buffer overflow, static and dynamic code and data tampering, ROP exploits were simulated and verified that the executable is able to withstand such attacks. The next steps to validate the results include porting the TrustZone code on actual Cortex-A15 based hardware and simulating various attack types and test the integrity of the system design and measure performance for complex security features such as DRM.

Appendix A

Bibliography

- [1] Counterpane Internet Security, Inc. <http://www.counterpane.com>
- [2] ePaynews - Mobile Commerce Statistics.
<http://www.epaynews.com/statistics/mcommstats.html> [checked 4/7/13]
- [3] Ravi, S., Raghunathan, A., & Chakradhar, S. (2004). Tamper resistance mechanisms for secure embedded systems. In *VLSI Design, 2004. Proceedings. 17th International Conference on* (pp. 605-611). IEEE.
- [4] Fiorin, L., Ferrante, A., Padarnitsas, K., & Carucci, S. (2010, October). Hardware-assisted security enhanced Linux in embedded systems: a proposal. In *Proceedings of the 5th Workshop on Embedded Systems Security* (p. 3). ACM.
- [5] Nokia, "Connecting Mobile Consumers and Merchants", Nokia White Papers, <http://whitepapers.zdnet.co.uk/0,39025945,60082474p-39000516q,00.htm>, 2005 [checked 4/7/13]
- [6] F-Secure Corporation, "Content Security at Hand: A White Paper on Handheld Device Security", www.fsecure.com/products/whitepapers/hhsecurity021122.pdf, 17/01/2005 [checked 4/7/13]
- [7] Hussin, W. H. W., Coulton, P., & Edwards, R. (2005, July). Mobile ticketing system employing TrustZone technology. In *Mobile Business, 2005. ICMB 2005. International Conference on* (pp. 651-654).

- [8] T. Alves and D. Felton. TrustZone: Integrated hardware and software security. ARM white paper, July 2004.
- [9] ARM, Fast model simulation reference,
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0423l/index.html>
- [10] Morimoto, R., & Guillet, J. (2008). *Windows server 2008 hyper-V unleashed*. Sams Publishing.
- [11] Ricci, L. and McGinness, L., “Embedded System Security. White Paper. Applied Data Systems” Columbia, USA, 2004
- [12] Gilmont, T., Legat, J. D., & Quisquater, J. J. (1999). Enhancing security in the memory management unit. In *EUROMICRO Conference, 1999. Proceedings. 25th (Vol. 1, pp. 449-456)*. IEEE.
- [13] Kai, T., Xin, X., & Guo, C. (2012, January). The Secure Boot of Embedded System Based on Mobile Trusted Module. In *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on*(pp. 1331-1334). IEEE.
- [14] IOS Jailbreaking, http://en.wikipedia.org/wiki/IOS_jailbreaking
- [15] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 1998.
- [16] B. Schneier, “Cryptographic Design Vulnerabilities,” In *IEEE Computer*, on vol. 31, pp. 29–33, Sept. 1998. IEEE.

- [17] Hadi Nahari, “Trusted Secure Embedded Linux - From Hardware Root Of Trust To Mandatory Access Control” MontaVista Software, Inc.
- [18] The IBM PCI Cryptographic Coprocessor. IBM Inc. (<http://www-3.ibm.com/security/cryptocards/>). [checked 4/7/13]
- [19] Messerges, T. S., Dabbish, E. A., & Sloan, R. H. (2002). Examining smart-card security under the threat of power analysis attacks. *Computers, IEEE Transactions on*, 51(5), 541-552.
- [20] Ferguson, N., Schneier, B., & Kohno, T. (2012). *Cryptography engineering: design principles and practical applications*. Wiley.
- [21] Open Virtualization, SDK and Linux OS, <http://www.openvirtualization.org/> [checked 4/7/13]
- [22] OpenSSL, Cryptographic Library, <http://www.openssl.org/> [checked 4/7/13]
- [23] Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D., & Kirda, E. (2010, December). G-Free: defeating return-oriented programming through gadget-less binaries. In *Proceedings of the 26th Annual Computer Security Applications Conference* (pp. 49-58). ACM.
- [24] Cowan, C., Wagle, F., Pu, C., Beattie, S., & Walpole, J. (2000). Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings* (Vol. 2, pp. 119-129). IEEE.
- [25] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., ... & Saxena, P. (2008). BitBlaze: A new approach to computer security via binary analysis. In *Information systems security* (pp. 1-25). Springer Berlin Heidelberg.

Appendix B

Simulation Environment

The Versatile Express (VE) components have been specifically developed to model in software some of the functionality of the VE hardware. A VE baseboard with Cortex A-15 processor was used for simulation. The following image shows the actual Versatile Express board.



The following tables summarize the memory map options:

Table A1: Versatile Express memory map

Peripheral	Modeled	Address range	Size
NOR FLASH0 (CS0)	Yes	0x00_00000000-0x00_03FFFFFF	64MB
Reserved	-	0x00_04000000-0x00_07FFFFFF	64MB
NOR FLASH0 alias (CS0)	Yes	0x00_08000000-0x00_0BFFFFFF	64MB
NOR FLASH1 (CS4)	Yes	0x00_0C000000-0x00_0FFFFFFF	64MB
Unused (CS5)	-	0x00_10000000-0x00_13FFFFFF	-
PSRAM (CS1) - unused	No	0x00_14000000-0x00_17FFFFFF	-
Peripherals (CS2) see ' '	Yes	0x00_18000000-0x00_1BFFFFFF	64MB
Peripherals (CS3) see	Yes	0x00_1C000000-0x00_1FFFFFFF	64MB
CoreSight and peripherals	No	0x00_20000000-0x00_2CFFFFFF ^a	-
Graphics space	No	0x00_2D000000-0x00_2D00FFFF	-
System SRAM	Yes	0x00_2E000000-0x00_2EFFFFFF	64KB
Ext AXI	No	0x00_2F000000-0x00_7FFFFFFF	-
4GB DRAM (in 32-bit address space) ^b	Yes	0x00_80000000-0x00_FFFFFFFF	2GB

Table A2: Secured memory options

Peripheral	Address range	Functionality with secure_memory enabled
NOR FLASH0 (CS0)	0x00_00000000-0x00_0001FFFF	Secure RO, aborts on non-secure accesses.
Reserved	0x00_04000000-0x00_0401FFFF	Secure SRAM, aborts on non-secure accesses.
NOR FLASH0 alias (CS0)	0x00_08000000-0x00_7DFFFFFF	Normal memory map, aborts on secure accesses.
Ext AXI	0x00_7e000000-0x00_7FFFFFFF	Secure DRAM, aborts on non-secure accesses.
4GB DRAM (in 32-bit address space)	0x00_80000000-0xFF_FFFFFFFF	Normal memory mpa, aborts on secure accesses.

Table A3: Peripheral Memory Map

Peripheral	Modeled	Address range	Size	GIC Int ^a
VRAM - aliased	Yes	0x00_18000000-0x00_19FFFFFF	32MB	-
Ethernet (SMSC 91C111)	Yes	0x00_1A000000-0x00_1AFFFFFF	16MB	47
USB - unused	No	0x00_1B000000-0x00_1BFFFFFF	16MB	-

Register spec can be found on the arm fast model website from the following link.

http://infocenter.arm.com/help/topic/com.arm.doc.dui0423m/DUI0423M_fast_model_rm.pdf

Appendix C

TrustZone API

Full TrustZone API which is made available by ARM can be found at the following link.

http://www.lcs.syr.edu/faculty/yin/teaching/CIS700-sp11/TrustZone_API_3.0_Specification.pdf