SUSTAINABLE TRUSTED COMPUTING

A Novel Approach for a Flexible and Secure Update of Cryptographic Engines on a Trusted Platform Module

Vom Fachbereich Informatik
der Technische Universität Darmstadt (D 17)
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieur (Dr.-Ing.)

von

Sunil Dath Kumar Malipatlolla
Master of Science (M.Sc.)

geboren in Pyararam, Indien

Referent: Prof. Dr.-Ing. Sorin A. Huss
Korreferent: Prof. Dr.-Ing. Abdulhadi Shoufan

Tag der Einreichung: 22.04.2013
Tag der mündlichen Prüfung: 09.07.2013

Darmstadt
April 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

[July 18, 2013 at 11:31]

This is the second version of the thesis entitled
"Sustainable Trusted Computing: A Novel Approach for a Flexible and Secure Update of Cryptographic Engines on a Trusted Platform Module".
The first version has been submitted on 25.09.2012 and has been withdrawn by the author on 05.11.2012

ACKNOWLEDGMENTS

The work presented in this thesis would have been possible if it were not for the help, support, guidance, and friendship of a number of individuals.

First, I would like to thank my advisor, Prof. Dr.-Ing. Sorin A. Huss, for his support and advice in the completion of this thesis. I would also like to thank my colleagues, especially Dr.-Ing. Abdulhadi Shoufan and Dipl.-Inform. Thomas Feller, for their fruitful research discussions. My gratitude also goes to my other colleagues at the ISS chair and the CASED institute.

I would like to thank CASED for funding my thesis.

I would also like to thank my parents, Rukminibai Malipatlolla and Yeshwanth Rao Malipatlolla, without their wisdom and support throughout my life, I would not have been able to accomplish all that I have. My special thanks go to my wife, Varsha Malipatlolla whose love and care were always with me.

ABSTRACT

Trusted computing is gaining an increasing acceptance in the industry and finding its way to cloud computing. With this penetration, the question arises whether the concept of hard-wired security modules will cope with the increasing sophistication and security requirements of future IT systems and the ever expanding threats and violations. So far, embedding cryptographic hardware engines into the Trusted Platform Module (TPM) has been regarded as a security feature. However, new developments in cryptanalysis, side-channel analysis, and the emergence of novel powerful computing systems, such as quantum computers, can render this approach useless. Given that, the question arises: Do we have to throw away all TPMs and lose the data protected by them, if someday a cryptographic engine on the TPM becomes insecure? To address this question, we present a novel architecture called Sustainable Trusted Platform Module (STPM), which guarantees a secure update of the TPM cryptographic engines without compromising the system's trustworthiness. The STPM architecture has been implemented as a proof-of-concept on top of a Xilinx Virtex-5 FPGA platform, demonstrating the test cases with an update of the fundamental hash and asymmetric engines of the TPM.

ZUSAMMENFASSUNG

Trusted Computing erhält zunehmend Akzeptanz in der industriellen Anwendung und findet auch seinen Weg in das Cloud Computing. Mit dieser Durchdringung stellt sich Frage ob das Konzept hardwarebasierter Sicherheitsmodule mit zunehmendem Reifegrad und steigenden Sicherheitsanforderungen zukünftiger IT-Systeme, sowie immer neuen Angriffsformen, noch immer geeignet ist. Bisher wurden kryptografische Hardwaremaschinen in Trusted Platfrom Modulen (TPM) als Sicherheitsfeature betrachtet. Neue Entwicklungen in der Kryptoanalyse, Side-Channel Analysen und die Entwicklung neuer Hochleistungsrechner wie Quantencomputer lassen dies jedoch fragwürdig erscheinen. Dabei stellt sich die folgende Frage: müssen wir alle TPMs wegwerfen und dabei all jene Daten verlieren die

auf ihnen basieren, wenn sich eines Tages herausstellt, dass die kryptografischen Maschinen dieser Module unsicher sind? Wir behandeln diese Frage indem wir einen neuen Ansatz, Nachhaltige Trusted Platfrom Module (STPM) genannt, präsentieren. Dieser Ansatz erlaubt eine sichere Aktualisierung der kryptografischen Maschine eines TPM ohne die Vertrauenswürdigkeit des Systems zu kompromitieren. Die STPM Architektur wurde prototypisch als proof-of-concept auf einer Xilinx Virtex-5 FPGA Plattform implementiert und demonstriert Testfälle für die Aktualisierung der fundamentalen Hashfunktionen und asymmetrischen Maschinen des TPM.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ABS	Automatic Breaking System
AES	Advanced Encryption Standard
AIK	Attestation Identity Key
AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application Specific Integrated Circuit
BIOS	Basic Input Output System
BRAM	Block Random Access Memory
CAD	Computer Aided Design
CAN	Controller Area Network
CE	Chip Enable
CLB	Configurable Logic Block
CMAC	Cipher-based Message Authentication Code
CMOS	Complementary Metal Oxide Semiconductor
COT	Chain-of-Trust
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check

CRTM Core Root of Trust for Measurement

DoS Denial-of-Service

DSA Digital Signature Algorithm

DSP Digital Signal Processor

ECC Elliptic Curve Cryptography

ECDSA Elliptic Curve Digital Signature Algorithm

ECU Electronic Control Unit

EDF Earliest Deadline First

EDK Embedded Development Kit

EEPROM Electrically Erasable Programmable Read Only Memory

EK Endorsement Key

EPROM Erasable Programmable Read Only Memory

FPGA Field Programmable Gate Array

HMAC Hash-based Message Authentication Code

ICAP Internal Configuration Access Port

I/O Input/Output

IP Intellectual Property

ISE Integrated Software Environment

IT Information Technology

JTAG Joint Test Action Group

LMB Local Memory Bus

LPC Low Pin Count

LUT Look-Up-Table

MAC Message Authentication Code

MDM Microblaze Debug Module

MITM Man-In-The-Middle

MMIO Memory Mapped Input Output

MTM Mobile Trusted Module

MUX Multiplexer

NGSCB Next-Generation Secure Computing Base

NIST National Institute of Standards and Technology

NVM Non-Volatile Memory

OIAP Object Independent Authorization Protocol

OS Operating System

OSAP Object Specific Authorization Protocol

PC Personal Computer

PCR Platform Configuration Register

PKC Public Key Cryptography

PR Partial Reconfiguration

PRF Pseudo Random Function

PRM Partially Reconfigurable Module

PROM Programmable Read Only Memory

PRR Partially Reconfigurable Region

PUF Physical Unclonable Function

RAM Random Access Memory

RML Reference Measurement List

RNG Random Number Generator

ROM Read Only Memory

ROT Root-of-Trust

RISC Reduced Instruction Set Computer

RSA Rivest Shamir Adleman

RTR Root-of-Trust for Reporting

RTS Root-of-Trust for Storage

SDR	Software Defined Radio
SFL	Security Function Layer
SHA	Secure Hash Algorithm
SHA-1	Secure Hash Algorithm-1
SHA-2	Secure Hash Algorithm-2
SKAP	Session Key Authorization Protocol
SMBus	System Management Bus
SML	Stored Measurement Log
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SRK	Storage Root Key
SSL	Security Service Layer
STC	Sustainable Trusted Computing
STPM	Sustainable Trusted Platform Module
TBB	Trusted Building Block
TC	Trusted Computing
TCG	Trusted Computing Group
TCL	TPM Command Layer
TCS	TCG Core Services
TCSI	TCG Core Services Interface
TDD	TPM Device Driver
TDDL	TCG Device Driver Library
TDDLI	TCG Device Driver Library Interface
THL	TPM Hardware Layer
TPM	Trusted Platform Module
TSP	TCG Service Provider

TSS TCG Software Stack

TTP Trusted Third Party

UART Universal Asynchronous Receiver/Transmitter

VHDL VHSIC Hardware Description Language

VM Virtual Machine

VPN Virtual Private Network

XMD Xilinx Microprocessor Debugger

1 INTRODUCTION

The advances in Internet technology have led to connecting more and more devices over a network and thus creating the possibility of an easy interaction between them. On the flip side, this interaction causes a compromised security and a loss of data on a device by discovering and exploiting its vulnerabilities. For example, one typical attack scenario in the area of personal computing is the online-banking, where the attacker may intercept the log-in details of the customer. Utilizing these details, the attacker would perform transactions though he is an unauthorized person. One such published work in this area is the attack carried out by the chaos computer club on the computer interface of the home banking system [10]. In case of enterprises using servers, the attacks are on the sensitive data such as company personnel, e-commerce, and others. This can be seen from a possible attack described by Spalka et al. in [84] against a certified signature application. Similarly in an automotive embedded system, an attack could be to make illegal changes to controllers within the system such as reducing the odometer reading to increase the car value [96]. Though a limited list of possible attack scenarios on different systems is given above, there are many such real world applications which are prone to regular attacks. Thus, there is a need to address the issue of data theft to assure the users and the organisations with the security of their personal data.

Information security is one such field that addresses the protection of individual data belonging to a certain device. The core concept of information security is to provide the *confidentiality*, *integrity*, and *authenticity* of the data, thereby avoiding an unauthorized access to it.

According to National Institute of Standards and Technology (NIST), they are defined as follows:

- *Confidentiality* protects the secrecy of the data by encryption such that the data may not be intercepted by the attacker. This is because the data being transferred on the communication channel is no more available in a plain text form but as a cipher.

- *Integrity* of the data is checked at both transmitting and receiving ends, is aimed at detecting any alteration of the data during its transfer on the communication channel. For this, the data being transferred is appended with its hash value which would be compared with the computed hash value at the receiver side. Thus, the hash values (digests) of the message are utilized to protect the integrity of the data.

- *Authenticity* ensures each entity at the end of the communication channel that it is talking to the right entity which it is intended to. Digital signatures and message authentication codes are the mathematical schemes that prove the authenticity of an entity.

Computer security is the specific branch of information security which provides various mechanisms, as illustrated below, for securing sensitive data on a Personal Computer (PC). The data on a computer may be protected by installing anti-virus software on it or by setting up a firewall in case that the computer is always connected to the Internet. The anti-virus is utilized to prevent, detect, and remove any malware threats such as computer viruses, computer worms, trojan horses, spyware, and others. However, some of these anti-virus software can reduce the computer's performance because they consume a lot of processing power. Whereas, a firewall may be software-based (as in most of the current Operating Systems (OSs)) or hardware-based (as in broadband routers) which protects the computer against threats from public Internet and keeps the network secured. Considering the case of protecting the data on a computer (not in a network) the firewall would be the software-based security solution included inside the OS. Furthermore, it is possible to improve the computer security by redesigning the operating systems or by changing programming methodologies.

However, all the aforementioned techniques provide only a software-based security to the user data, which is weaker than any available hardware-based protection mechanism. This is because the software-based solutions always require a running OS and which can easily be modified compared to hardware. This has been confirmed from experience of security trends in the smartcard world which applies equally to host systems such as PC platforms. A smartcard has its own OS for communicating with a server via a card reader. The smartcard also has its

own file system, in which the OS can protect files by means of encryption, passwords or PIN codes. Although in the military and government sector there have been some attempts to develop purely software-based high-security systems, even there the access to the appropriate hardware is greatly limited. This leads to an additional disadvantage that the functionality and flexibility of the OS is generally very restricted. Thus, to provide a comprehensive defense against the security threats and to provide a trusted and tamper-proof security, more than one form of security approach is required to be included in the system.

In this context, changes to the design of hardware is one useful method among many for improving security of the platform. While hardware changes to a device are not a prerequisite for increased security, they are undeniably helpful. For example, adding an additional dedicated security module to the motherboard of a PC could be one such hardware modification. This dedicated security module may provide a way to securely store the sensitive cryptographic data such as the private keys of the user thereby defending the data protected by those keys [7]. One such technology to enhance the security of the PCs through hardware changes made to the underlying architecture is the Trusted Computing (TC) [6]. The other initiative is the Microsoft's OS project, referred to as Next-Generation Secure Computing Base (NGSCB) [3]. This project specifies software changes that take advantage of the security benefits made available by a planned new PC hardware design. The other less well-known such projects are by processor vendors such as the Trusted Execution Technology (TXT) from Intel [2] and the Secure Execution Mode (SEM) from AMD [1]. These projects provide the hardware support for all the major features needed by TC and NGSCB initiatives. However, in this thesis, only details about the TC approach are given because it is utilized as the standard technology specification for this work.

1.1 TRUSTED COMPUTING

Trusted Computing has been developed and promoted by the Trusted Computing Group (TCG) [6], which is an international de facto standards body with a consortium of large number of Information Technology (IT) enterprises and hardware manufacturers such as AMD, Intel, IBM, DELL, HP, Microsoft, and others. Trusted Computing Group creates specifications and

guidelines for developing hardware modules that can be used to augment the security in end-user systems. These specifications define trusted modules for devices such as PCs and other devices, trusted infrastructure requirements, as well as Application Programming Interfaces (APIs) and protocols necessary to operate a trusted environment. Further, without such standard security procedures and shared specifications, it is not possible for components of the trusted environment to inter-operate and trusted computing applications cannot be implemented to work on all platforms. Though a proprietary solution for building a trusted environment may be possible, it cannot ensure global interoperability and is not capable of providing a comparable level of assurance due to more limited access to cryptographic and security expertise and reduced availability for a rigorous review process. Furthermore, from the point of view of cryptography, for interoperability with the other elements of the platform, other platforms, and infrastructure, it is necessary for trusted modules to be able to use the same cryptographic algorithms. Although standard published algorithms may have weaknesses, these algorithms are thoroughly tested and are gradually replaced or improved when vulnerabilities are discovered which is not true in the case of proprietary algorithms. In the above mentioned context, the standards published by the TCG are utilized for building trusted platforms that incorporate the TC technology. The concept of trusted computing has been described in many publications ranging from the Orange Book [74] through to more recent material that describe these ideas within the context of contemporary developments in computer security [45, 83].

According to TCG, "a trusted platform is the one which behaves in an expected manner for the intended purpose" [91]. This means that any entity that interacts with such a trusted platform can be given some degree of assurance that the platform (a system or a computer) will behave in the way that entity expects it to. Providing assurance of expected behavior does not in itself provide any security, instead to achieve that, the entity relying on this assurance still has to ascertain that the expected behavior is indeed secure. This assurance may be achieved by utilizing *remote attestation* procedure, which is one of the main features of the TC technology. However, assuming that the relying entity is satisfied that the expected behavior is secure then the entity may be assured that any data given to the system is kept confidential or that no malware is running on the plat-

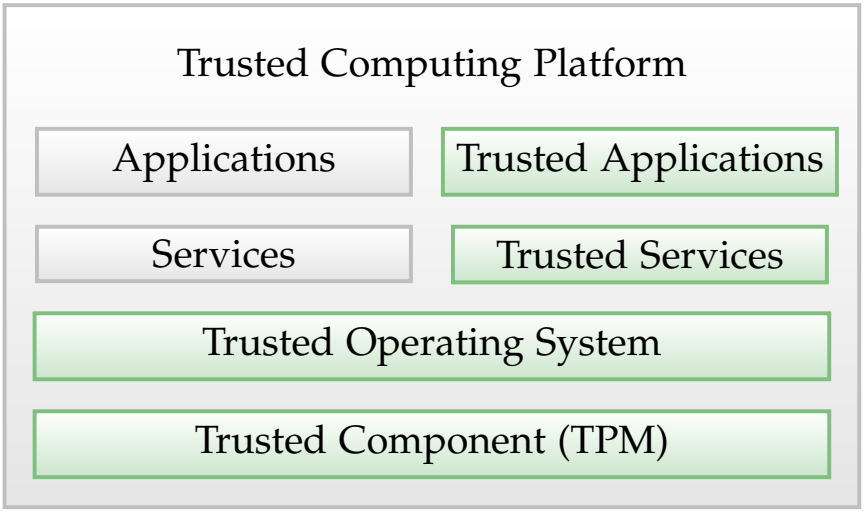

Figure 1.1: Trusted Computing Platform [30]

form. Though there are many ways to design trusted computing platforms that allow statements about expected behavior to be made, the approach taken by the TCG is to have some physically secure trusted component, that can be used as a foundation upon which trust in the rest of the system can be built. Such a trusted component to provide this foundation of trust is the Trusted Platform Module (TPM) [8], which is specified by the TCG. The TPM is implemented as a tamper-proof integrated circuit and is affixed to the platform, usually on a printed circuit board (in specific motherboard) containing a more powerful processor (i.e., main processor) capable of running software applications. Thus, the TPM can be used as the trust anchor for higher level processes that run on the main processor.

A trusted computing platform [22] utilizing the TPM as the trusted component is depicted in Figure 1.1. Such a platform defines the services required for the implementation of the trusted computing concept extensions to the hardware of the system. This includes for example the use of new components on the motherboard, the installation of expansion cards or the adaptation of existing components such as the Central Processing Unit (CPU) or the chipset. These enhancements may include the adaptation of the closely related software with the hardware i.e., the firmware. The trusted operating system with its central component as secure operating system kernel (trusted kernel), provides the interface for trusted applications and trusted services.

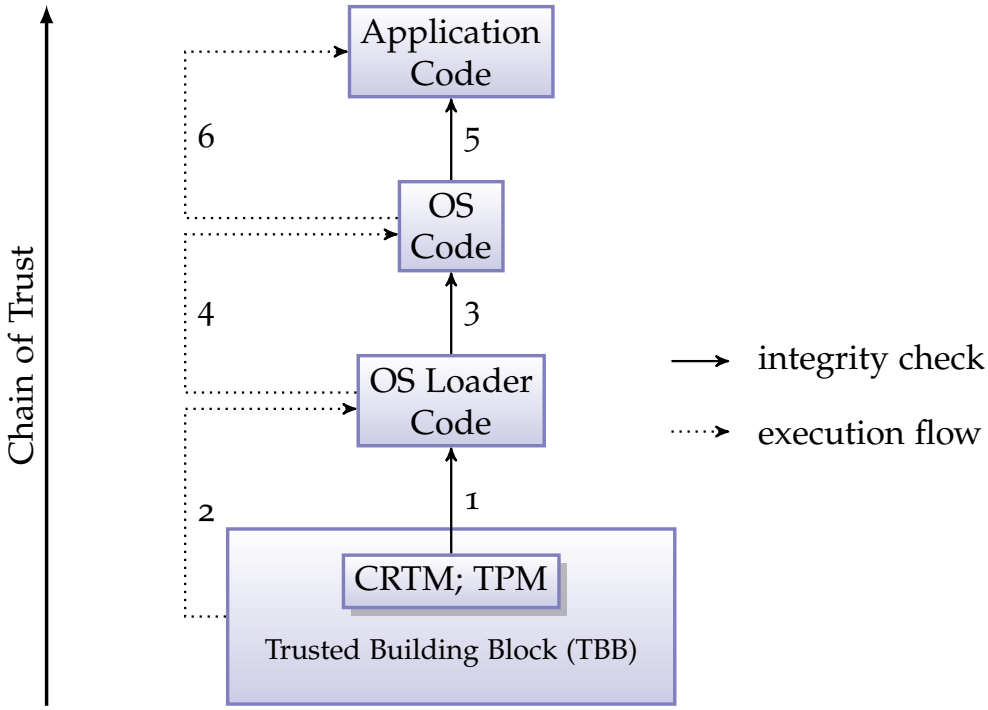

Figure 1.2: Chain-of-Trust [30]

In contrast to achieving the platform security by means of additional encryption or anti-virus software at a higher level, the TCG approach begins at the lowest level of the platform. The approach relies on the secure measurement of a certain level in the boot process of a system before transferring the execution control to the next level. Each level in the process, called as an entity, cryptographically measures the next entity, stores the measured value inside the registers of the hardware module and then transfers the control to it. Further, a log entry containing the utilized register number, stored value, and a log message is made. This process of measuring and passing control creates a Chain-of-Trust (COT) [22], as depicted in Figure 1.2, because every predecessor entity in the chain must be trusted before trusting the current entity. Now, an important question arises, i.e., how to trust the first entity in the chain or in other words who measures the first entity? For this, the TCG says: "to build a trusted platform, the first and foremost entity in the COT must always be trusted because its misbehavior can not be detected". This entity which is always trusted is called as the trust anchor or the Root-of-Trust (ROT) in the TCG terminology. The TPM is one of the specifications of the TCG,

which forms the ROT inside a PC to provide the trusted computing features and to build trusted platforms. The cryptographic measure computed while building the COT is a machine readable value, called the " hash value", which represents the state (configuration) of the platform. These hash values are stored inside the hardware module, in this case the TPM, to protect them from being tampered with. Later, these values are utilized to attest (trustworthily report) the state of the system to an external entity such as a remote challenger, when needed to prove the system integrity.

In specific, Figure 1.2 illustrates the process of building a trustworthy PC utilizing a TPM that extends the trust from the lowest layer up to the applications. Though the TPM forms the theoretical ROT, for implementation purposes it is the Core Root of Trust for Measurement (CRTM), a small part of Basic Input Output System (BIOS), which acts as the actual ROT. Further, the CRTM combined together with the TPM forms the Trusted Building Block (TBB) of the system. According to TCG specification, the TBB must always be trusted to build a trusted platform. With reference to Figure 1.2, the COT begins with the measurement of the BIOS by the CRTM before passing control to the boot loader code as shown by arrow 1 (integrity check) and arrow 2 (execution flow) respectively. However, the arrow depicting the action of storing the *integrity measurement* inside the TPM and creating a log entry for stored value are not depicted in Figure 1.2, but are performed at every stage of the COT. These integrity check and execution flow actions continue till all the entities in the chain are measured and their hash values are stored securely inside the TPM. Upon request, these hash values together with the log entries can later be used to report the trustworthiness of a system to a remote challenger and thus allowing a secure remote access (c.f. Section 1.2).

The TCG standard specification for TPM is largely based on the experiences from high-security smartcards and their applications. Further, the important parts of the smartcard architecture and security characteristics have been consistently adopted while designing the TPM. Similar to the way in which the smartcard's cryptographic mechanisms are utilized to protect the sensitive and confidential personal data, these mechanisms can also be utilized in the TPM. These functions when utilized in a platform embedded with a TPM, not only ensure the integrity of that platform but also protect the user data by providing a secure processing environment. Thus, the TPM provides au-

thentication and accreditation of the platform in addition to securely storing the critical secrets such as cryptographic keys and digital certificates.

For this task, it accordingly includes, in addition to secure firmware principles and components, particular hardware requirements of the type well-known from design principles for high-security crypto smartcards. However, inserting a secure TPM into the PC platform along with any other standard PC modules does not prevent intelligent attacks performed with hardware debugging and analysis tools. In essence, the TPM increases the resistance and security level of a platform, i.e., by storing sensitive data inside it, but it must be taken into account that a complete security of the platform is not achieved with it. Considering that the ability of a modern computing platform (such as a PC or a laptop) to protect sensitive information through software alone is limited, the TPM provides a better and more secure solution. This is because in a PC without TPM, software cannot verify its integrity and, even if software modifications can be detected locally, there is no mechanism for a remote user to be assured that the software is trustworthy. Whereas when using a TPM, the logs on the software modifications are stored inside the hardware on the TPM, which can later be provided to the remote challenger to decide upon the trustworthiness of the system. Thereby, the TPM provides the ability for a computing system to run applications more securely and to perform electronic transactions & communications more safely.

In essence, the TPM is really nothing more than a cryptographic co-processor tightly coupled to the CPU that requires software support from the BIOS and the host operating system. The TPM provides two classes of functionality: integrity protection and trusted storage. Both these functionalities require a basis for their security guarantees for which they use trusted root certificates. The security guarantees are then provided through the application of these functionalities as the system boots and subsequently operates.

However, there are certain potential downsides of the TCG specification that come from the aspects of un-changeable trusted root and inability to provide the desired degree of user privacy. The first aspect when coupled with the inability to disable the cryptographic measurement capability, would prevent users from running an operating system of their choice. Thus, it could also prevent the use of free operating systems because

the OS kernel would have to be signed by an entity which is a descendant of the trusted root. To cope with aspect of privacy, the TCG implements a trusted third party system where the user presents their identity and receives an anonymous credential from an anonymity certification authority (the trusted third party). However, there are two major problems with this approach as illustrated below [17]. The first is that if a user requests several anonymous credentials, the trusted third party can still link all of the anonymous credentials to the user because of their knowledge of the user's identity. The second problem is that if the trusted third party ever conspires with a true name certification authority then it is easy to attach the user's real identity with their anonymous identities by matching public keys. Though the TC technology has its own positive and negative aspects, regular updates are made by TCG to improve it towards a user friendly technology.

1.2 TRUSTED PLATFORM MODULE

Trusted Platform Module is a tamper-resistant device with limited computational power, such as a cryptographic smartcard, that has a processor (not programmable by the end-user), a limited amount of non-volatile storage, hardware accelerators for performing cryptographic operations, and secure Input/Output (I/O) interfaces. A TPM is securely bound to the motherboard of a platform, whereas a smartcard is associated with a specific user and is portable to be used across multiple systems. In a PC the TPM is integrated into the system over the Low Pin Count (LPC) bus, which is utilized for communication between the central processor and the TPM as depicted in Figure 1.3. In general, high-speed devices are connected to the north bridge and low-speed devices to the south bridge. The LPC bus is a 4-bit wide bus operating at a frequency of 33.3 MHz, connects the low bandwidth devices such as the BIOS chip and the TPM chip to the south bridge. This bus is introduced by Intel and is a substitute for industry standard architecture (ISA) bus that is 16-bit wide and operating at a frequency of 8.33 MHz. As aforementioned, the CRTM is the part of BIOS, which acts as the actual ROT is also depicted in Figure 1.3. This description is at an higher level depicting how the TPM is integrated into the computer system (i.e., a PC). The internal components of the TPM itself are briefly described in the sequel.

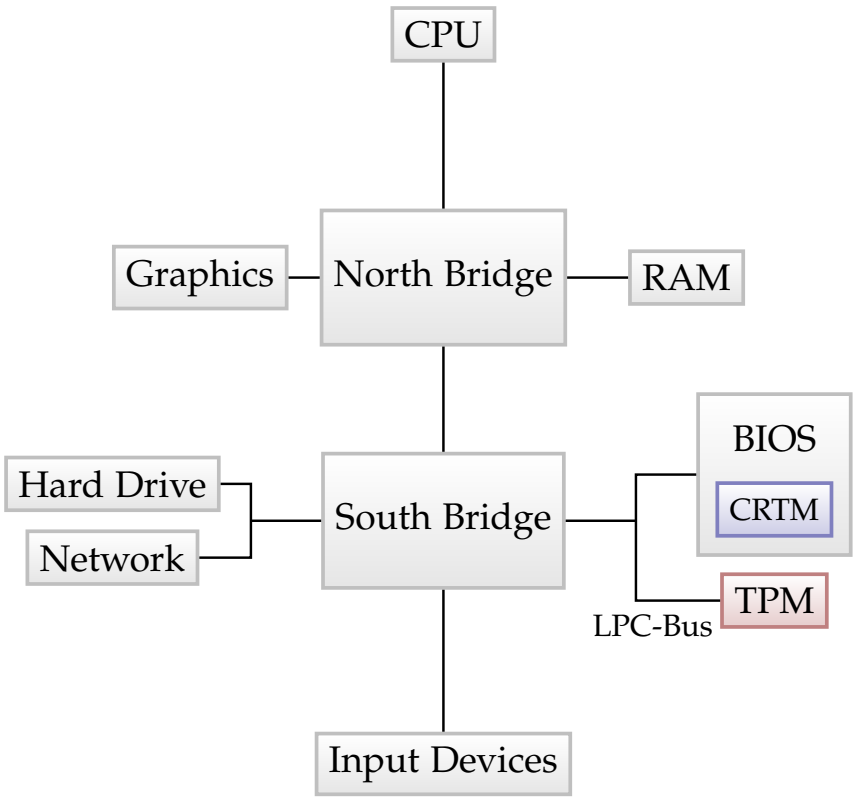

Figure 1.3: PC Reference Architecture of the TCG [30]

Conventionally, TPM is a microcontroller-based chip with hard-wired engines for various cryptographic schemes such as Rivest Shamir Adleman (RSA), Secure Hash Algorithm-1 (SHA-1), and Hash-based Message Authentication Code (HMAC) as depicted in Figure 1.4 [22]. The commands given to the TPM are processed by an execution engine, which is a microcontroller, with the help of Random Access Memory (RAM) (a working memory) and Read Only Memory (ROM) (memory with stored program code) on it. The cryptographic memory units of the TPM comprises of two storage units i.e., one non-volatile and the other volatile. The non-volatile unit stores the sensitive cryptographic data such as keys and credentials that are unique to the platform and the volatile unit is utilized for storing the system state and temporary keys. In essence, utilizing the above features, the TPM provides a hardware-based security to the system (user) artifacts such as data, certificates, passwords, and other cryptographic keys.

An useful application of the TPM in a general-purpose computing system, i.e., hard drive encryption, is described in the

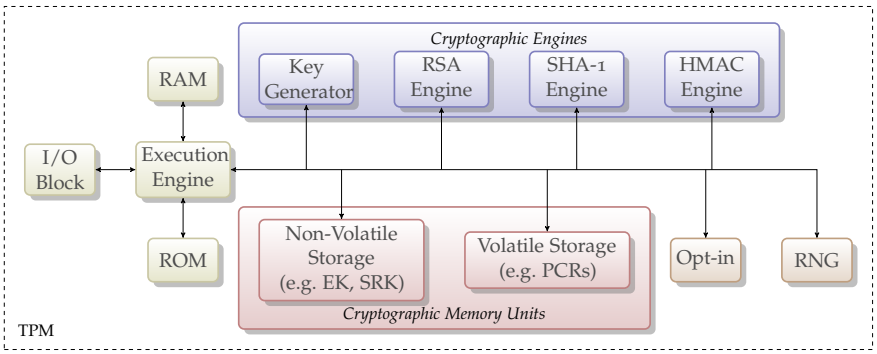

Figure 1.4: Conventional TPM Architecture [30]

following. In a PC without TPM, encrypted data and encryption keys are usually stored on the same hard drive. With this, an attacker can mount this hard drive onto another computer to disclose the keys and thus the data. In contrast, a PC enabled with TPM stores the encryption keys on the TPM and thus prevents an unauthorized access to the data. From this, it may be said that the security of modern computing platforms is enhanced via the use of a trusted hardware i.e., the TPM. There are the firms such as Atmel, Broadcom, ST Microelectronics, Infineon, Sinosun and Intel which manufacture the TPM hardware. On the other hand, several vendors in the market such as Acer, Asus, Dell, LG, Fujitsu, HP, Lenovo, Samsung, Sony, and Toshiba provide TPM integration on their devices.

A trusted computing architecture utilizing a TPM provides a trusted system utilizing the following key concepts [7]:

MEMORY CURTAINING

Memory curtaining refers to a strong, hardware-enforced memory isolation feature to prevent programs (by executing them in their own address spaces) from being able to read or write one another's memory. In its current design, an intruder or malicious code can often read or alter sensitive data in a PC's memory whereas in the trusted computing design, even the OS should not have access to curtained memory. Thus, an intruder who gains control of the OS itself would not be able to interfere with program's secure memory. Although memory isolation can be achieved in software, this requires some combination of rewriting operating systems, device drivers, and possibly even application software. Implementing this feature in hardware instead permits greater backwards compati-

bility with existing software and reduces the quantity of software which must be rewritten.

SECURE INPUT AND OUTPUT

The aim of secure input and output feature is to address the threats posed by keyloggers and screen-grabbers, software used by snoops and intruders to spy on computer user's activities. A keylogger records what you type, and a screen-grabber records what is displayed on the screen. To avoid this, the secure input and output feature provides a secure hardware path from the keyboard to an application and from the application back to the screen. This is achieved by encrypting the I/O stream right up to the point that the output device outputs data or from the point that the input device inputs data. No other software running on the same PC will be able to determine what the user typed, or how the application responded. At the same time, secure input and output will allow programs to determine whether their input is provided by a physically present user, as distinct from another program impersonating a user.

SEALED STORAGE

Sealed storage addresses a major PC security failing i.e., the inability of a PC to securely store cryptographic keys. Customarily, the keys and passwords that protect private documents or accounts are stored locally on the computer's hard drive, alongside the documents themselves. This may be compared to leaving the combination to a safe in the same room with the safe itself. In practice, intruders who break into a computer can frequently copy decryption and signing keys from that computer's hard drive.

To address this issue, the sealed storage idea was conceptualized which generates keys based, in part, on the identity of the software requesting to use them and, in part, on the identity of the computer on which that software is running. The result is that the keys themselves need not be stored on the hard drive but can be generated whenever they are needed, provided that authorized software tries to generate them on an authorized machine. If a program other than the program that originally encrypted, or sealed, private data should attempt to decrypt, or unseal, that data, the attempt is guaranteed to fail. Similarly, if the data is copied in encrypted form to a different machine,

attempts to decrypt it will continue to be unsuccessful. Thus, sealed storage represents a clever solution to a previously intractable key storage problem. Sealed storage combined together with memory curtaining and secure input and output feature ensures that the user data on a system can only be read by that specific computer and only by the particular software with which it is created. Even if an intruder or a virus deliberately alters the encryption software, it will no longer be able to decrypt the data and thus the data is protected.

REMOTE ATTESTATION

Remote attestation is the most significant of the four major features, which broadly aims to detect the unauthorized changes to software. If an attacker has replaced one of the applications or a part of the OS on the platform with a maliciously altered version, it would be detected by utilizing remote attestation feature. This is because the attestation is remote, i.e., the other party with whom the platform interacts will be able to detect the alteration and thus can avoid sending sensitive data to a compromised system. However, the current TCG approach to the remote attestation is flawed because it conspicuously fails to distinguish between applications that protect computer owners against an attack and applications that protect a computer against its owner itself. In effect, the computer's owner is sometimes treated as just another attacker or adversary who must be prevented from breaking in and altering the computer's software.

Remote attestation works by generating, in hardware, a cryptographic certificate attesting to the identity of the software currently running on a PC. However, there is no determination of whether the software is good or bad, or whether it is compromised or not compromised. The identity in here is represented by a cryptographic hash, which allows different programs to be distinguished from one another or to detect changes in their code. This certificate may, at the PC user's request, be provided to any remote party, and in principle has the effect of proving to that party that the machine is using expected and unaltered software. If the software on the machine has been altered, the certificate generated will reflect this and then the re-

mote party can decide whether to interact with the user or not.

TRUSTED THIRD PARTY

One of the main obstacles the TCG technology had to overcome was how to maintain the anonymity while still providing a ”trusted platform". The main objective of obtaining a ”trusted mode" is how *system B*, with whom *System A* may be communicating, can trust that the latter is running an un-tampered hardware and software. For this, the latter has to inform the former that it is using ”registered" and ”safe" software and hardware, thereby potentially uniquely identifying itself. This might not be a problem where one party wishes to be identified by the other party, e.g., during banking transactions over the Internet. But in many other types of communicating activities the anonymity provided by the system may be a mandatory requirement. The TCG acknowledges this, and has developed a process of attaining such anonymity but at the same time assuring the other party that it is communicating with a ”trusted party". This is achieved by an entity referred to as ”trusted third party", which acts as an intermediary between the two parties and is utilized during the remote attestation process.

Each of these above concepts are desired to enhance the computer security, because each may be used by appropriate software to prevent or mitigate real attacks currently performed against PCs. Thus, a PC with hardware support for these concepts can provide security guarantees that might be difficult to offer without hardware support. Although trusted computing technology can not prevent computer security holes altogether, it seeks to contain and limit the damage that can result from a particular flaw. For instance, it should not be possible for a coding flaw in one application (like a web browser) to be abused to copy or alter data from a different application (like a word processor). This sort of isolation and containment approach is an important area of computer security research and is used in many different approaches to computer security, including promising techniques outside of trusted computing. However, to utilize the new capabilities of the PC obtained through TC, a software support is necessary which is already specified by TCG and is referred to as TCG Software Stack (TSS) [90]. This software stack provides an abstract interface to the applications on

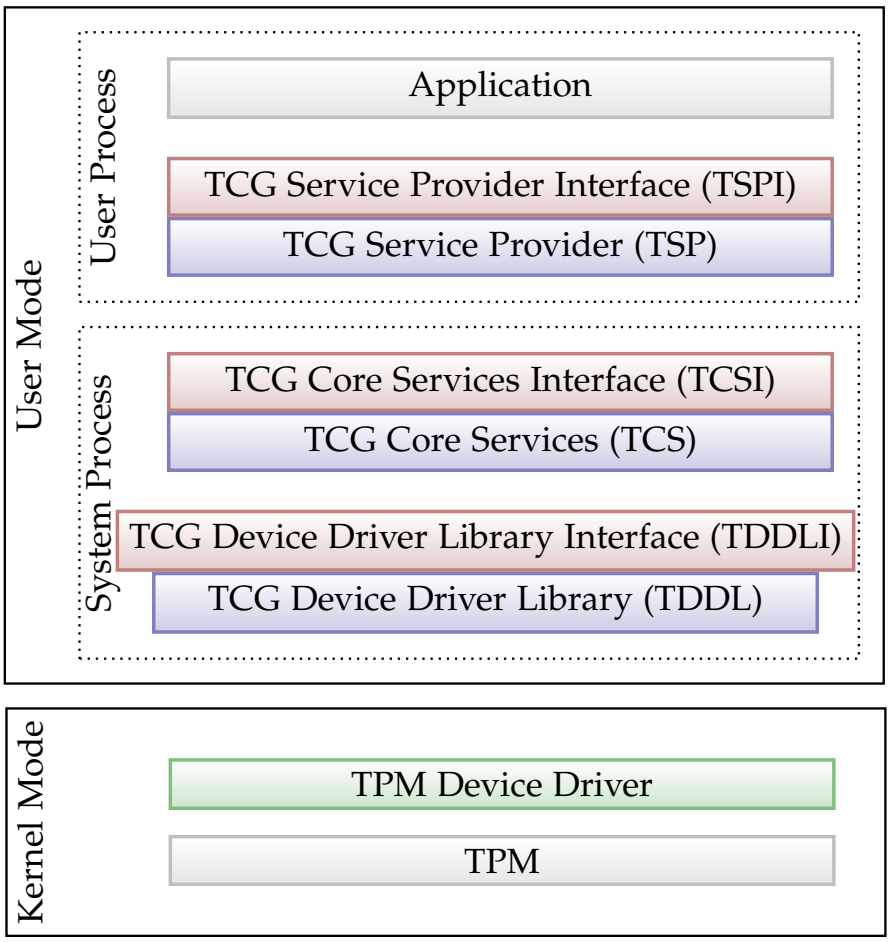

Figure 1.5: Trusted Software Stack [30]

the system such that they can easily access the functionalities of the TPM.

The multi-layer framework of the TSS [22], depicted in Figure 1.5, is illustrated in the following:

TPM DEVICE DRIVER
: TPM Device Driver (TDD) is not a part of the TSS, but part of the (extended) operating system kernel. It must be addressed with the TCG Device Driver Library (TDDL) and the corresponding port, the TCG Device Driver Library Interface (TDDLI), from the TSS components. This is supported by current operating systems (Linux kernel version 2.6.18, Windows Vista, Windows 7, and Windows 8) but previously (until TPM version 1.1) for each manufacturer a special driver had to be used.

TCG DEVICE DRIVER LIBRARY

The TDDL is available in user-mode of the OS which controls the transition between kernel-mode and user-mode. Through an implementation of the interface in user-mode, it is neutralized from operating system, such that different TSS implementations can communicate with each TPM. The TDDLI allows an easy implementation of the TPM simulator (see [86]). The simultaneous TPM access is controlled by higher layers, thus only one instance of the TDDL is required.

TCG CORE SERVICES

The TCG Core Services (TCS) are part of the user-mode OS services, which provide the interface, TCG Core Services Interface (TCSI), for the service provider and implement the four basic services of TSS including: The context manager which controls the concurrent access to the TPM. The key & credential manager implementing the key management and secure storage of key material outside of the TPM. The event manager that managing the entries in the event log in the form of cryptographic measurements and access to the corresponding Platform Configuration Registers (PCRs). Responsibility for serialization, synchronization and transmission of commands to the TPM.

TCG SERVICE PROVIDER

The TCG Service Provider (TSP) is also a part of the OS service, which provides functions that go beyond the TPM functionality. An important component of the TSP is the one with cryptographic functions and interfaces that needs only slight adjustments to be used for existing applications.

Utilizing the aforementioned hardware (TPM) and software (TSS) specifications from TCG, a hardware-based root of trust is provided to a given platform. The trust anchor or the ROT being the critical component of the security architecture, its integrity compromise would jeopardize the security of the entire system. Thus, the TCG demands its implementation in hardware, because it is safer than an equivalent implementation in software. Further, to uniquely identify a system, a forgery or a copy of the system must be ruled out, a requirement which can only be ensured by utilizing a hardware-based ROT.

Some of the applications where TPMs can be employed are described below:

PERSONAL COMPUTING

In PCs, the TPM may be used for full disk encryption and password protection. Full disk encryption applications, such as the BitLocker Drive Encryption of Microsoft's operating systems i.e., Windows Vista and Windows 7, use the TPM chip in conjunction with the included disk encryption software named BitLocker [41]. The Bitlocker technology is used to protect the keys used to encrypt the computer's hard disk and to provide integrity authentication for a trusted boot-up process. System artifacts such as cryptographic keys, passwords, and digital certificates are more secure against software attacks and physical thefts as the authentication mechanism is shifted to hardware when using a TPM.

VIRTUALIZATION

Virtualization technology provides concepts and methods for sharing or combining of hardware resources between computer systems, especially in servers [101]. This enables to run multiple operating system instances, referred as Virtual Machines (VMs) or guests, on a single real computer called host machine. Virtualization is gaining importance in academia and industry because it increases the overall utilization of resources and provides economic benefits. This can be seen from the fact that Intel and AMD have added virtualization support for their processor architectures to support multiple VMs on a single entity [94, 12]. To provide the VMs with trusted computing features, the host machine utilizes a TPM chip which enhances the security of the individual VMs and thus the whole system together.

MOBILE COMPUTING

Consider a mobile platform scenario in which the device manufacturer, cellular service provider, application service provider, and the user himself act as different stakeholders. Here, the task of the mobile platform is to provide a trustworthy attestation of the current state of each of their applications to the respective stakeholder where a TPM may be deployed.

CLOUD COMPUTING

In cloud computing, the applications and the data are no longer on the local computer or local data centers, but in

a central computing server. In such a scenario, the user must trust that his applications and data on the remote server are safe in which case a TPM may be used for a trustworthy operation.

Though there are many other application fields for TPMs, the application scenario in this work refers to a general-purpose computing system i.e., a PC.

1.3 PROBLEM DEFINITION

With the ability to provide the aforementioned security features, the TPM is gaining an increasing acceptance in the industry and in personal computing which led to its wide range deployment. According to market analysis, already over 300 million TCG-compliant TPM chips are included in many business laptop and desktop systems [28]. In contrast with this penetration, the question arises whether the concept of hard-wired cryptographic engines will cope with the increasing sophistication and security requirements of future systems and the ever expanding threats and violations. This is because, it is well-known that cryptographic schemes, also those embedded in TPMs, have always been subject to persistent cryptanalysis and recently to side-channel analysis either by malicious attackers or by the research community. Given that, the question arises: Do we have to throw away all the TPMs and lose the data protected by them, if someday a cryptographic engine on the TPM becomes insecure? Consider that either the hash engine (SHA-1) or the asymmetric engine (RSA) on the TPM is compromised, i.e., found to be no more secure. This implies that all the security functions such as remote attestation and sealed storage, as detailed before, that rely on these engines can not be trusted anymore.

There exist a few publications in the literature which indicate that this threat of insecure cryptographic algorithms is indeed true. For instance, Wang et al. [98] showed the collision search attacks for the SHA-1 algorithm and Finke et al. [42] conducted a side-channel attack on the RSA key generator. Further, in [23], Bruschi et al. have presented a replay attack during the execution of the TPM authorization protocol that compromises the correct behavior of the trusted platform. Also, Sadeghi et al. have tested several TPM chips for compliance to TCG specifications and were able to find weaknesses with those chips as described in [80]. Additionally, considering the scale of de-

ployment in the market as mentioned before, replacing a compromised TPM with a new one every time certain engine gets compromised is certainly not economically feasible.

Considering these and other threats and violations, and following the general recommendation regarding the necessity of updating cryptographic algorithms, e.g., those published by NIST [71] and proposed by Preneel [76], the idea of hard-wiring the TPM security engines began to be questioned. This can be seen from the specification of the next generation TPM (called TPM.next) [88] by TCG. In particular, this specification allows the replacement of cryptographic algorithms in case of a compromise or even for boosting the TPM performance. In spite of this agreement on their necessity, technical solutions for the update of TPM's cryptographic engines have not been proposed in the literature, so far.

1.4 THESIS CONTRIBUTION AND ORGANIZATION

To address the issue of compromised (or broken) cryptographic engines on a TPM, a novel concept called Sustainable Trusted Computing (STC), is presented in this thesis. The core idea of this concept is to design a novel architecture, referred to as Sustainable Trusted Platform Module (STPM), which can support a flexible and secure update of the cryptographic engines on the TPM and then to re-establish the trust in the system after an update. To achieve this, the STPM comprises of static and reconfigurable parts in its architecture. The static part is equipped with the un-interruptedly executing components such as the execution engine (for TPM command execution), an update algorithm (for securely loading new cryptographic engines), and an interaction between them. In contrast, the reconfigurable part of the architecture consists of the locations for loading the new cryptographic engines as needed. Further, essential components such as non-volatile and volatile memories for cryptographic keys and platform state storage respectively, are also integrated into the architecture. To evaluate the STPM design, a proof-of-concept implementation is performed utilizing a Field Programmable Gate Array (FPGA) as its base component. The considered FPGA supports a special feature called Partial Reconfiguration (PR), utilizing which the FPGA is divided into static and reconfigurable parts as required by the STPM design. Each of these parts contain the necessary components for performing a successful update and to provide all the necessary TPM functionalities. To

show the applicability of the STPM and/or its individual components, two different scenarios are chosen and evaluated.

The rest of this thesis is organized as follows:

- A thorough architectural and functional description of all the building blocks of the current state-of-the-art TPM, specified by the TCG, is given in Chapter 2. Further, an in-depth analysis of the effect of every compromised cryptographic engine of the TPM on its functionalities is carried out and corresponding countermeasures are discussed.
- Based on this analysis, a set of functional and non-functional requirements that lead to a set of architectural specifications and consequently to designing a novel generic updatable TPM architecture, i.e., the STPM, is described in Chapter 3. Additionally, the major components of the STPM architecture which provide all the security functions with reference to conventional TPM are detailed. Considering that the major functionality of the STPM architecture is to provide a secure update and re-establish the trust in the system, an analysis of the same is performed utilizing two fundamental cryptographic engines on it.
- A proof-of-concept implementation of the STPM architecture utilizing available components in the market is detailed in Chapter 4. In addition to illustrating the resource consumption of the STPM design on the target platform, an update of the fundamental cryptographic engines utilizing the implemented design is also detailed.
- Two real world applications, i.e., Intellectual Property (IP) protection in embedded systems and security in real-time systems, that may utilize the features of the STPM are evaluated in Chapter 5.
- Chapter 6 concludes the thesis and highlights some future research directions.

2

TRUSTED PLATFORM MODULE

Though the TCG's specifications for TPM version 1.2 are extensive and complex ranging over 700 pages, we concentrate here only on the concepts that are relevant to this thesis. For this, first the building blocks of state-of-the-art TPM architecture are detailed before explaining the security functions provided by it [56]. Then a thorough analysis of the security level of each building block is carried out to illustrate the effect on corresponding functionalities due to the possible attacks on the building blocks.

The fundamental feature of the TPM is to provide a hardware-based security to the sensitive data on a platform. This form of security ensures that the information stored in hardware is better protected from external software attacks. Using this feature, a variety of applications storing secrets on a TPM can be developed, which make it much harder to access information on computing devices without proper authorization (e.g., if the device was stolen). Further, if the configuration of the platform has changed as a result of unauthorized activities, access to data and secrets can be denied and sealed off using these applications. However, it is noteworthy to state that the TPM cannot control the software that is running on a PC. The TPM can only store the pre run-time configuration parameters, but it is the other applications that determine and implement policies associated with this information.

For example, processes that need to secure secrets, such as digital signing, can be made more secure with a TPM. Mission critical applications requiring greater security, such as secure email or secure document management, can offer a greater level of protection when using a TPM. If at boot time it is determined that a PC is not trustworthy because of unexpected changes in configuration, access to highly secure applications can be blocked until the issue is remedied. However, this is possible only if a policy has been set up that requires such an action. Additionally, with a TPM, one can be more certain that artifacts necessary to sign secure email messages have not been affected by software attacks. Further, with the use of *remote attestation*, other platforms in the trusted network can make a determina-

tion, to which extent they can trust information from another PC. Attestation or any other TPM function does not transmit personal information of the user of the platform. These capabilities can improve security in many areas of computing, including e-commerce, citizen-to-government applications, confidential government communications and many other fields where greater security is required. Also, hardware-based security can improve protection for Virtual Private Network (VPN), wireless networks, file encryption (as in Microsoft's BitLocker) and password/credentials management.

2.1 TPM BUILDING BLOCKS

The essential building blocks of today's TPMs that provide the fundamental security functions of the TCG specification, are depicted in Figure 1.4. These various blocks are hard-wired and may be categorized into three different groups: cryptographic engines, cryptographic memory units, and a processing unit. The cryptographic engines unit comprises of various cryptographic algorithms that perform the operations required by the security functions. In order to perform these operations, the TPM requires some cryptographic keys and platform state, which are stored in different memories of the cryptographic memory unit. To supplement these, the processing unit comprises of an execution engine (a microcontroller) and the firmware to execute the commands given to the TPM.

In the following, each of the block and its functionality is detailed:

KEY GENERATOR

This block is an asymmetric key generator which utilizes the RSA algorithm [79] to generate the keys required by the TPM. The generated keys are public/private key pairs of size 2048-bit, which is the minimum recommended key size by the TCG.

RSA ENGINE

This is an asymmetric engine that performs the encryption and decryption operations on the data (and keys) to be protected by the TPM. Furthermore, this engine is utilized for digital signature generation/verification, as required during remote attestation process. Together with the Storage Root Key (SRK) and the key generator, the RSA engine builds the TPM key hierarchy.

SHA-1 ENGINE

This is a hash engine utilized to compute the digest of the existing hardware or a loaded software on the platform, that represents the state of the platform. The SHA-1 engine makes use of 160-bit SHA-1 hash algorithm [43] thus, the produced digest is also of size 160 bits (20 bytes). Considering that the TPM is not a cryptographic accelerator, there are no TCG specified minimum throughput requirements for the engines of the TPM.

HMAC ENGINE

HMAC is a Message Authentication Code (MAC), which utilizes a cryptographic hash function and a symmetric key mainly for testing the authenticity of messages. This engine utilizes the HMAC algorithm that must be implemented as specified in RFC 2104 [58] standardization. The HMAC engine is required to provide authorization to all the TPM commands and the related data during a command execution. The TCG specification specifies the use of SHA-1 in HMAC, resulting in a key length of 20 bytes and a block size of 64 bytes. The HMAC function is only available internal to the TPM.

RANDOM NUMBER GENERATOR

This block is the source of entropy which generates hardware-based true random numbers to be utilized by the TPM for key generation and nonces. The Random Number Generator (RNG) consists of a state-machine that accepts and mixes unpredictable data and a post-processor that uses a one-way function (e.g., SHA-1).

OPT-IN

This component is utilized to enable/disable the TPM. However, the TPM is disabled by default and the owner may enable it via the Opt-in.

NON-VOLATILE STORAGE

This block stores the sensitive cryptographic data i.e., the Endorsement Key (EK) and the SRK of the TPM. Whereas the EK uniquely identifies the given platform, the SRK is utilized to take ownership and to build the key hierarchy of the TPM. The other sensitive information stored inside this memory is the TPM permanent flags and the owner authorization secret (described later). These flags deter-

mine the configuration of the TPM such as its working state and optionally also affect its functionality.

VOLATILE STORAGE
: The significant part of the volatile storage are the PCRs which store the hash values of the entities measured by the TPM during the boot process of a system. Other part of this storage being the key slot which stores the temporarily generated keys before loading them on and off the TPM. Further, the volatile storage holds the handles (abstract references) for the TPM keys and for the authorized sessions.

EXECUTION ENGINE
: This component is an 8-bit microcontroller and together with the working memory RAM, it executes the TPM commands by utilizing the program code available in ROM.

I/O BLOCK
: The I/O block is an interface between the central processor on the system and the TPM blocks, that provides a secure flow of data between them. For this, it performs encoding and decoding of information passed over internal and external bus.

The TPM utilizes symmetric encryption unit such as Advanced Encryption Standard (AES), only for internal operations such as encrypting the authentication data, i.e., passwords and PINs to secure data to be communicated or stored outside of it. There are additional components inside the TPM such as the monotonic counters and sensors which are needed to prevent replay attacks and detect physical attacks (tampering), respectively. Thus, the security of the whole system is based on the protection and secrecy of the TPM's cryptographic system, especially against reading out or manipulation of the key material. In a typical computer system, the TPM is enclosed in a secure package and is affixed to the motherboard of that device. Figure 2.1 depicts such a binding of a security chip (Atmel TPM) to the motherboard (Asus) of a PC to enhance the security of the system.

The TCG specification defines two generic portions of the TPM: shielded locations and protected capabilities. The data inside a shielded location is protected against the interference from an outside exposure. The non-volatile storage with EK and SRK data is a shielded location of the TPM. In contrast, the protected

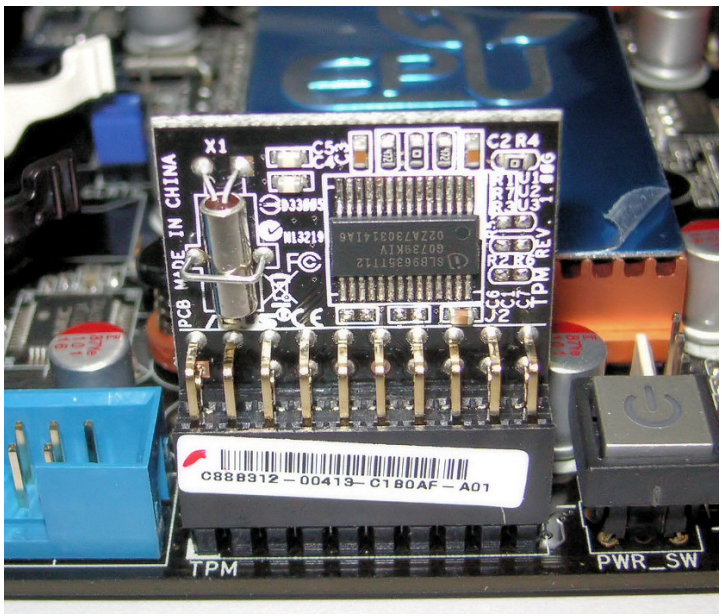

Figure 2.1: TPM chip on a motherboard [44]

capability is a function that can access (read/write) a shielded location. The function "take ownership" is one such protected capability which creates a new root key (i.e., SRK) of the TPM key hierarchy. Further, a correct operation of the protected capabilities is necessary for a trusted operation of the TCG subsystem. Both shielded locations and protected capabilities are implemented in hardware and therefore are resistant against software attacks.

Some of the TPM features are explained in the following. The TPM is only a platform component but not a complete platform by itself. However, it is becoming a common component on many business desktop and laptop systems. The TPM is not an active component i.e., it only responds to commands from the processor but does not initiate any interrupts or such operations. Further, the TPM cannot alter the execution flow of the system such as booting or execution of applications. In summary, the TPM is a shielded and encapsulated chip with a controlled interface to the external environment.

The life cycle of a TPM is depicted in Figure 2.2. First, the TPM is in *Power off* state, if it is off or in a power saving state. After power-up or a reset the TPM goes into the initialization state. The reset is performed by executing the *TPM_Init* command, which is not transmitted from a software component but is embedded in the hardware of the TPM. Then according to specifications, the BIOS sends the command *TPM_Startup* within the

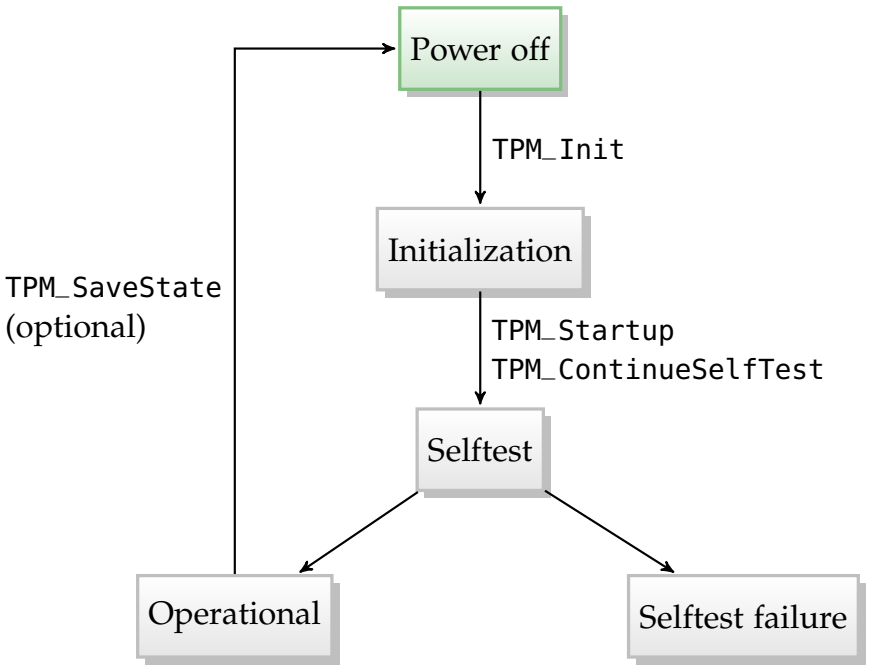

Figure 2.2: TPM Life Cycle [30]

CRTM, utilizing which the TPM transits into the *Selftest* state. From this state, the TPM either moves into an *Operational* state if the *Selftest* passes else moves into *Selftest failure* state. Once operational, the TPM may save the state of the platform and then moves into *Power off* state.

However, the operating states of the TPM may be changed for security reasons by the platform owner and critical state changes can be accomplished only with physical presence. There are three possible pairs of states for the TPM: Disabled/Enabled, Deactivated/Activated, and Owned/Unowned. State transitions can be achieved by executing various TPM commands in each state. An example of such a *TPM* command is the *TPM_DisablePubekRead*, which mandates the detection of the physical presence and prevents the reading of the EK without the owner password.

2.2 TPM KEYS

An important feature of the TPM is to provide and manage the key hierarchy. It can internally generate RSA keys, for which the key generator and the RNG are used. Through the generation of cryptographic keys inside the TPM, it is possible that the private part of a key does not have to leave the TPM in a clear form and therefore is only known to the TPM. Key pairs can be used to

digitally sign (signing key) and encrypt the data storage (storage keys). All the keys generated by the TPM are the RSA key pairs with public and private part for each key in a given key pair. Though the typical RSA key lengths can be 512, 1024 or 2048 bits, the TCG recommends only the use of 2048 bit.

The keys generated by the TPM have different attributes that indicate the allowed usage and the level of protection. Every TPM has a set of foundational keys, i.e., the EK and the SRK, and all the other keys are generated using these keys. Due to the fact that the TPM has limited memory on-board, the newly generated keys must be stored outside of it on a mass storage device (e.g., hard disk). Additionally, key slots are required for temporary storage of the keys while loading them in and out of the TPM. However, the newly generated keys, referred to as child keys, can not be stored in a clear text form thus every generated child key pair is protected by an associated parent key pair. For this, the private part of the child key pair is encrypted by the public part of its parent key before the former key leaves the TPM. To utilize the child key for a certain operation, it must first be loaded into the TPM for decrypting with the private part of its parent key. This loading of the keys into an external memory creates a hierarchy, referred to as TPM key hierarchy, as depicted in Figure 2.3. The illustrated key hierarchy, which is typical inside a PC, depicts only the required components of the TPM for building it.

The different types of keys specified by the TCG may be described as follows:

MIGRATABLE AND NON-MIGRATABLE KEYS

Non-Migratable keys such as EK and SRK are bound to a given platform and are not allowed to leave the TPM thus protecting the sensitive data encrypted by them. In contrast, migratable keys are the ones which can be migrated to a different platform but only under the authorization of the TPM owner. Migratable keys may be needed in case that there is a change in the TPM version but the user is not willing to update the key hierarchy.

STORAGE KEYS

These keys are utilized to encrypt data or other keys, thus storing the other keys securely outside of the TPM. This encryption and storage of keys, referred to as key wrapping, builds the key hierarchy. Further, storage keys are at the top of the key hierarchy, which can only generate the

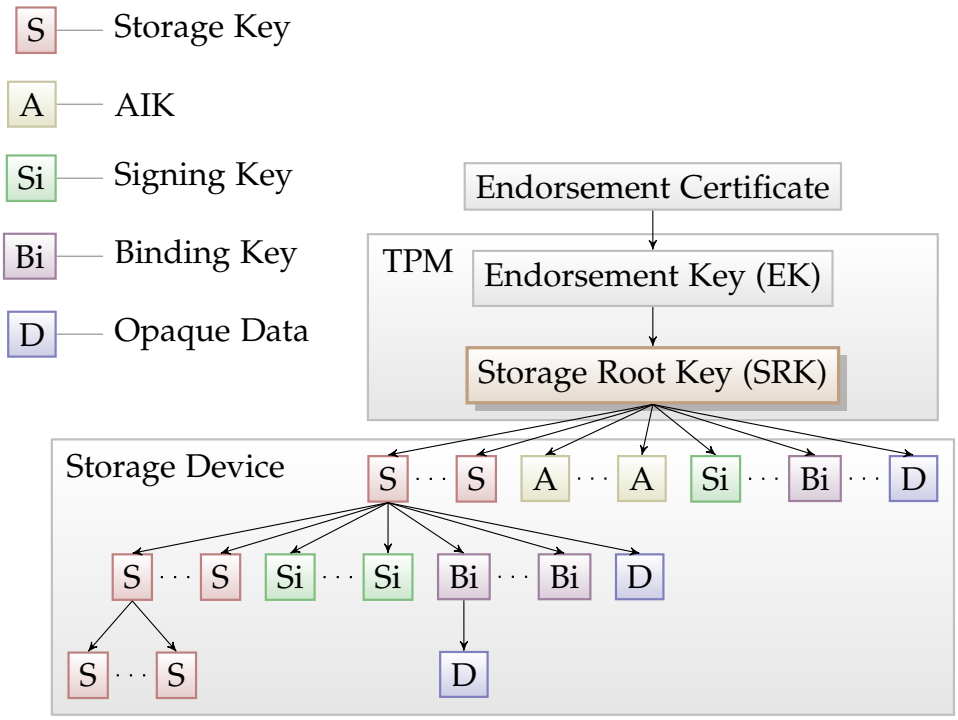

Figure 2.3: Typical TPM Key Hierarchy inside a PC [30]

child keys. A special storage key is the SRK, explained below, forms the Root-of-Trust for Storage (RTS) of the TPM.

SIGNING KEYS

Signing keys are utilized to digitally sign the data. These keys may be either tied to a given platform or may be migrated to a new platform.

BINDING KEYS

In general, these are the symmetric keys that encrypt large amounts of data outside the TPM called as hybrid encryption. Binding keys can be migrated to other platforms.

ENDORSEMENT KEY

The EK is a public/private key pair, generated usually by the manufacturer or vendor of the TPM as part of the manufacturing process. The entity that creates the EK must provide a credential (certificate) certifying that this key pair is valid and was generated in the specified manner. Typically, this entity is the manufacturer of the TPM. The EK uniquely identifies a TPM and thus the platform, which cause privacy concerns with this key. EK is utilized to at-

test the authenticity of the values produced by the TPM. The EK is non-migratable and can only be used in carefully controlled ways, such as creating identity keys as detailed below. This key forms the Root-of-Trust for Reporting (RTR) of the TPM.

ATTESTATION IDENTITY KEY

Attestation Identity Keys (AIKs) are non-migratable keypairs that are essentially aliases to the EK. Due to privacy concerns, the EK is not used to sign any information originating from a TPM. As a non-migratable key, the private portion of an AIK never leaves the TPM in plaintext form and is used only for signing data originated by the TPM, such as PCR values during *remote attestation*. The EK is statistically unique, whereas multiple AIKs (acting as pseudonyms for the EK) may be generated by the owner of the TPM thus mitigating privacy concerns.

STORAGE ROOT KEY

The SRK is a non-migratable keypair that is generated internally to the TPM while taking ownership of the platform. The private part of this key never leaves the TPM. This key is at the root of the TPM key hierarchy and is used for secure storage of all the other keys below it. Further, when setting up a TPM owner, the owner authorization secret which is a 160-bit long hash representation of the owner's password is stored in the TPM. This password is encrypted with the public part of the EK and therefore can only be decrypted by the TPM.

In order to use a stored key, its private part must first be loaded into the TPM for decryption by its private parent key. If the child key to be used is not the SRK, then all the storage keys in the path from SRK to the key to be used must be loaded recursively. The key is then stored in a key slot and a reference (key handle) is addressed. However, the number of key slots is limited, therefore the TCG recommended temporary paging is to be used for a storage medium, which is managed by the key cache manager. This manager may be part of the TSS, as well as managing the key hierarchy, i.e., the assignment of child keys to parent keys.

As a self-protection measure, the TPM implements two ROTs: Root-of-Trust for Reporting and Root-of-Trust for Storage. The RTR is the foundation for the unambiguous identification of the

platform and the RTS is the basis for secure data storage. Their integrity is protected by the capability of TPM to resist software and hardware attacks. Further, the TCG prompted a number of protective mechanisms such as the safety-critical information (e.g., EK) should not be read via the contacts of the TPM microchip (e.g., by pin probing). The TPM must be using a solder joint or integrated with other components (e.g., South Bridge) and shall be secured to the platform. An attempt to remove EK and SRK, i.e., tampering with the TPM device, is detected and countermeasures to prevent the compromise are provided. For example, voltage, clock frequency and other aspects of the TPM's operating environment may be monitored for signs of tampering.

2.3 TPM SECURITY FUNCTIONS

The TPM provides four fundamental security functions: *integrity measurement*, *remote attestation*, *binding*, and *sealing*.

INTEGRITY MEASUREMENT, STORAGE, AND REPORTING

With this function the platform configuration and the processes running on it are measured cryptographically by determining their hash values. These values are stored inside the PCRs and further a history of events is kept in the Stored Measurement Log (SML). There are at least 16 PCRs in a TPM and a PCR is not written directly but by a process called "extend operation" illustrated below. The SML resides in the hard drive (outside, and not protected by a TPM). To provide *integrity measurement* functionality, the TPM utilizes the SHA-1 engine.

The following notation is given for describing the *integrity measurement* process:

entity	An executable application, a configuration file or a data file in a PC platform
$SHA-1$	Hash algorithm
$\mathit{measurement}$	Measured hash value of the entity by utilizing $SHA-1$
PCR	Register for storing the measurement

The "PCR extend" operation is described as follows:

$$PCR \Leftarrow SHA-1(PCR||\mathit{measurement}) \tag{2.1}$$

A new measurement value is concatenated ("||" symbol) with the current PCR value and then hashed by SHA-1 such that the result will be stored as a new value of the PCR. The advantage of the extend operation is that it is infeasible to find two different measurement values such that when they are extended return the same value. Further, the operation preserves the order in which entities' measurements were extended (for example, extending entity A then entity B results in a different value than extending B then A). Additionally, the operation allows unlimited number of measurements to be stored in a PCR, because the result is always a 160-bit value.

In a COT, if an entity B is loaded after the entity A, then B is measured by A before passing control to B. Further, the measurement is stored into a PCR, by an extend operation, and also into the SML. The benefit of following this order is that B can not lie about its existence i.e., the fact that it had been loaded and run. Assuming that if B is a malicious program, it tries to avoid being detected by removing its measurement in the SML. However, B can not remove its measurement from the PCR, because the PCR is protected at the hardware level. Additionally, no part of the system can write directly to the PCR, except the TPM, and it is computationally infeasible to find another program whose hash value is the same as B. Thus, the *integrity measurement* mechanism guarantees an unforgeable record of all the entities that have been loaded, with the help of the SML and the TPM. However, in most systems, the PCR and the SML keep only a single measurement of each loaded program and does not take into account subsequent loads of the same program, as well as number of times it is loaded. Though taking the measurement every time a program loaded reflects the live configuration of the platform, it slows the speed of operation of the system and affects its scalability.

REMOTE ATTESTATION

The trustworthy reporting of the platform configuration to a remote challenger is called *remote attestation*. The TPM provides a set of PCR values representing the system state and the SML to the challenger. These values are signed using the AIK private key. The remote party is able to compare the measurement results with reference values

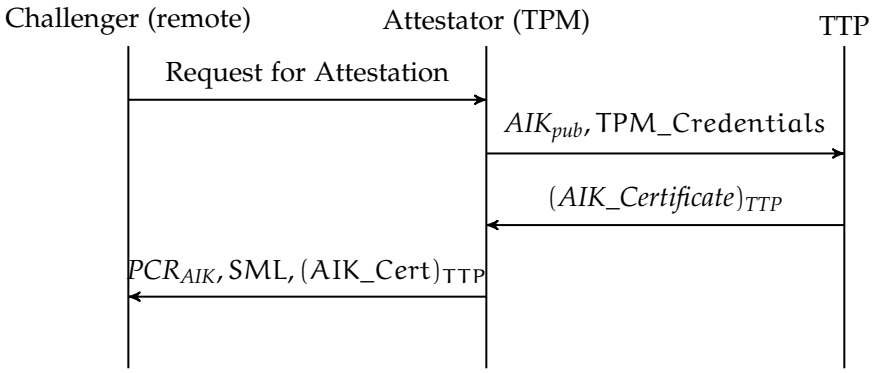

Figure 2.4: Remote Attestation Protocol [8]

that indicate the trusted platform configurations. These reference values are usually listed in a public Reference Measurement List (RML). For providing the *remote attestation* functionality, the TPM utilizes both RSA and SHA-1 engines.

The *remote attestation* protocol utilizing a Trusted Third Party (TTP), i.e., a privacy certification authority, is depicted in Figure 2.4. The challenger requests for an attestation to the system with TPM on it i.e., the attestator. For this, the attestator generates an AIK and sends its public part to a TTP along with the TPM credentials. In turn, the TTP generates a certificate for the AIK and sends it back to the attestator. The attestator signs the PCR values with the private part of the AIK and sends them to the challenger along with the obtained certificate and the SML. The challenger verifies the obtained signature and recomputes the hash values using the SML to decide upon the trustworthiness of the attestator.

BINDING

This functionality binds the data (usually the cryptographic keys) to a given platform. It utilizes asymmetric encryption to store the keys in the key hierarchy managed by the particular TPM. To bind the data, RSA engine of the TPM and a migratable key are utilized. The migratable key utilized in *binding* gives the flexibility to decrypt the user's encrypted data by another TPM.

$$\text{Binding} \Rightarrow \text{Encrypt}_{\text{Migratable\_Key}}(\text{Data}) \tag{2.2}$$

Security Function	Involved Cryptographic Engines
Integrity Measurement	SHA-1
Remote Attestation	RSA, SHA-1
Binding	RSA
Sealing	RSA, SHA-1

Table 2.1: TPM Security Functions Mapped to TPM Building Blocks

SEALING

Sealing extends the *binding* functionality by tying it to the platform state i.e., the data along with certain set of PCR values are encrypted together. Therefore, the data which has been encrypted at some platform state, can only be decrypted if the platform is exactly in the same state. In addition to RSA engine, the TPM utilizes a non-migratable key and a set of requested PCR values to seal the data.

$$\mathtt{Sealing} \Rightarrow \mathtt{Encrypt}_{\mathtt{Non\_Migratable\_Key}}(\mathtt{Data}||\mathtt{PCR\_Values}) \tag{2.3}$$

To decrypt the sealed data, one must be running the same TPM, have the key, and the current PCR value has to match with the value used in the *sealing* process. For example, consider a document is sealed with a TPM-generated non-migratable key, and PCR values indicating that Microsoft Word and Symantec antivirus must have been loaded. In order to read the contents of this document, other users must have access to the key, be using Microsoft Word and Symantec antivirus software in the same TPM otherwise, the data remains sealed.

All the aforementioned security functions rely on one or more cryptographic engines as detailed in Table 2.1. Therefore, a compromise of one of these engines leads to a damage of at least one of the security functions. For instance, if the SHA-1 engine is attacked, neither of the security functions will be available except for the *binding* functionality. Similarly, if RSA algorithm is compromised, except for *integrity measurement* no security functionality is reliable. Note that the HMAC engine is not listed in Table 2.1 as it does not directly support any of the main secu-

Parameter	Length	Description	Byte	Byte	Byte	Byte
1	2	Authorization Tag	TPM_TAG			
2	4	Parameter Size	UINT32			
3	4	Ordinal	TPM_COMMAND_CODE			
4 . . . n	. . .	*Payload (opt.)*	. . .			

Table 2.2: Components of a TPM-Command [30]

rity functions. However, HMAC engine is essential for data and command authorization and is therefore intrinsically essential for all the security functions. Given that the HMAC utilizes the same hash engine already available on the TPM, a compromised SHA-1 implies that all the security functions are compromised, as commands can not be securely issued anymore.

2.4 TPM COMMANDS

The TPM is controlled by *TPM-commands* that are communicated generally over the LPC bus. The *TPM-command* is sent as an input to the TPM and after processing the TPM sends back the *TPM-response*. In both cases, i.e., the command and the response, there are sequences of bytes with defined structure which are transmitted over the LPC bus from most significant to least significant byte. It is clear that the TPM is designed as a passive component that can not intervene on its own into the operation of a system. The general components of a *TPM-command* are illustrated in Table 2.2. The authorization tag indicates the form of authorization required by the given command. The size parameter specifies the total number of bytes that make up the command and the ordinal specifies the command to which it is actually referring to. Optionally, depending on the command type one or more parameters of various lengths (referred to as the payload) such as handles, nonces, hashes, data values are also required by a command.

The *TPM-response* that is sent in response by the TPM, is composed similar, as shown in Table 2.3. Instead of ordinals as aforementioned, in the return code there are status signals which indicate the successful or failed execution of a *TPM-command*. Complete details of all the commands and their responses are detailed in the TPM specification [8].

However, safety-critical commands to the TPM such as the *TPM_TakeOwnership* may be given only with prior authoriza-

Parameter	Length	Description	Byte	Byte	Byte	Byte
1	2	Authorization Tag	TPM_TAG			
2	4	Parameter Size	UINT32			
3	4	Return Code	TPM_RESULT			
4 . . . n	. . .	*Payload (opt.)*	. . .			

Table 2.3: Components of a TPM-Response [30]

tion via a secure channel. For this, a command validation protocol is required for the authorization of the caller and for ensuring the integrity of the authenticated channel and thus the transmitted messages. The TCG specifies two essential command validation protocols, as described in the sequel, which utilize an implementation of the HMAC [8].

OIAP
: The Object Independent Authorization Protocol (OIAP) allows the development of a communication channel, wherein the generated random number (nonce) may be utilized for accessing all the TPM managed objects such as the managed keys. The channel is valid indefinitely unless it is terminated by the user or by the TPM. The corresponding command for this protocol is the *TPM_OIAP*.

OSAP
: In contrast, the Object Specific Authorization Protocol (OSAP) allows the development of a communication channel, where the generated nonce may be utilized for accessing only a single TPM managed object. The nonce authorizes the subject to execute several commands that target a single object. The advantage is that for operations on the object only once a password must be specified. The one-time password can be optionally used to encrypt the transmitted messages. The corresponding command for this protocol is the *TPM_OSAP*.

Although most TPM commands allow both command-validation protocols, OIAP is preferable because it creates a session for authorization of multiple objects. However, OSAP must be used when authorization data (e.g., passwords that are utilized for key management) is to be reset. These protocols initiate a rolling-nonces scheme such that at every step of the protocol, both the user and the TPM reference previously sent nonces and introduce a new nonce. Nonces newly introduced by the user are

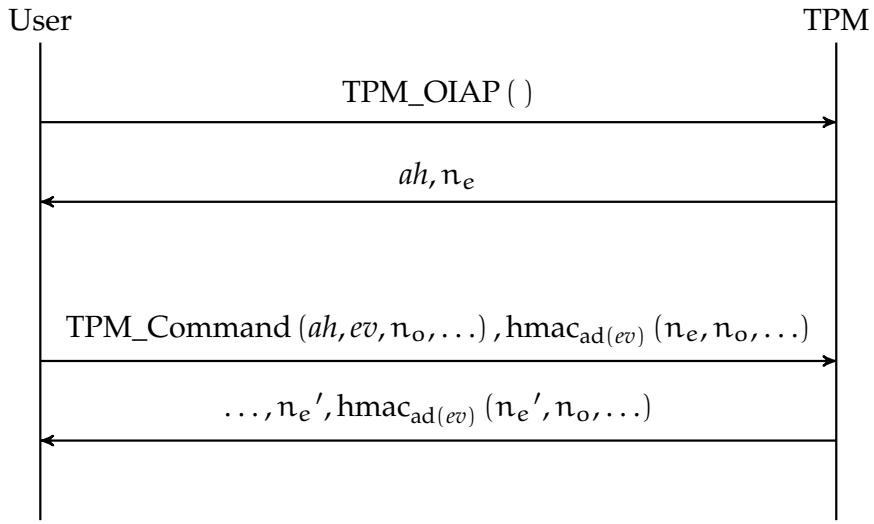

Figure 2.5: OIAP[30]

denoted as odd and those introduced by the TPM are denoted as even. The generalized operation of OIAP and OSAP protocols is depicted in Figure 2.5 and Figure 2.6, respectively.

To establish a communication channel, the user executes the *TPM_OIAP* command to the TPM with no additional parameters. In response, the TPM sends a authhandle, *ah*, back as the identification of the session and the first nonce n_e. A subsequent *TPM-command* from the user refers to an entity (an object) with identity *ev* and the session that is identified by *ah*. At the same time the new nonce n_o is introduced and the authorization is performed via the HMAC computation on the two exchanged nonces n_e and n_o. The authorization data ad (ev) serves as the key for the HMAC computation. In response, the TPM provides a new nonce n_e' and the outcome of the HMAC over the last and current nonces. Further communication in the session is carried out by a new nonce and the created HMAC over the last and current nonces (i.e., rolling-nonces). Similar to OIAP, the OSAP also includes a rolling-nonces scheme. However, the key for the HMAC computation for the communication will be calculated as a session key *S*, which is again an HMAC calculation using the key ad (ev) with previously exchanged OSAP nonces.

Both OIAP and OSAP protocols are vulnerable to off-line dictionary and impersonation attacks, as shown in [25]. Thus, it is suggested by the TCG that these protocols may be replaced or supplemented by a new protocol, referred to as Session Key

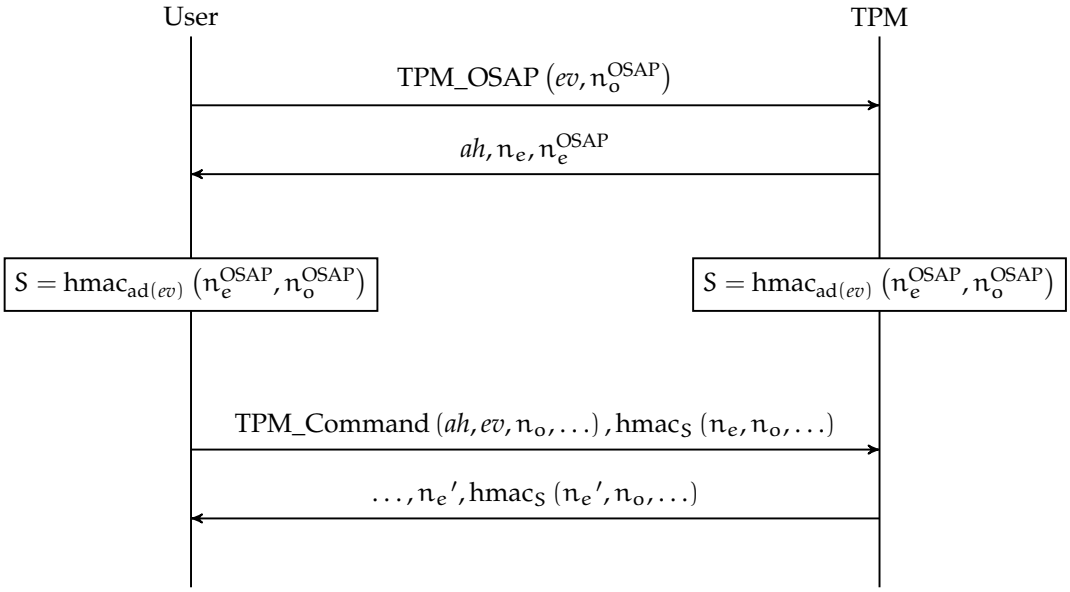

Figure 2.6: OSAP[30]

Authorization Protocol (SKAP), which is proposed by Chen et al. in [26]. In addition to the above protocols, there exist another form of access control which is the verification of physical access (physical proof of presence) on the system. This requires, at system start-up, an interaction must take place with the platform in order to execute a secure command. For this, the TCG provides two options: First, a corresponding control signal can be applied to the TPM chip, which is set by a special key combination or a jumper on the motherboard. Secondly, a particular *TPM-command* (such as *TSC_PhysicalPresence*), signaled by a software component that produces physical presence request. In the latter case, the component must have previously interacted with the user, and the TCG recommends this implementation as part of the CRTM.

2.5 THREATS TO THE TPM AND THEIR IMPLICATIONS

For the purpose of analyzing the effect of cryptographic attacks on the TPM security functions, a 4-layer abstraction model for trusted computing is introduced as depicted in Figure 2.7.

- The Security Function Layer (SFL) accommodates the previously described TPM security functions and represents the most abstract view. Trusted computing also specifies several auxiliary security functions which are required for

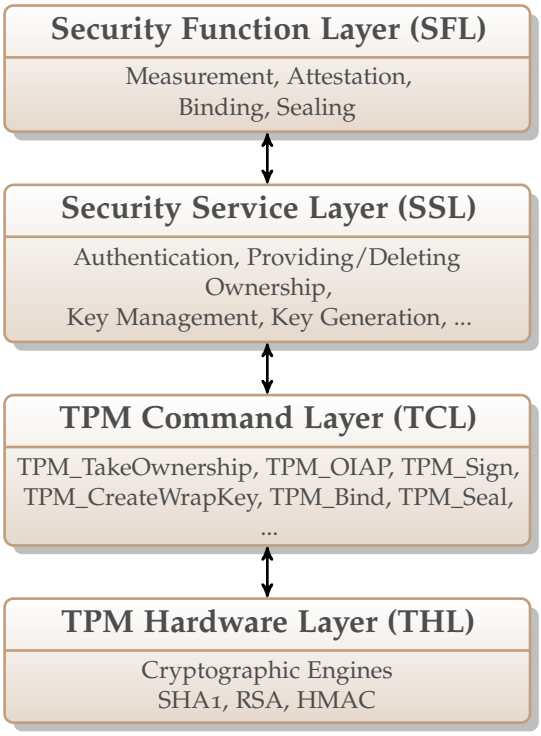

Figure 2.7: Layered Abstraction Model for Trusted Computing[11]

correct and secure usage of the TPM. Actions to take ownership, to delete an owner, and to authenticate the TPM commands are examples of these functions which we refer to as security services. These services are accommodated on a dedicated layer called the Security Service Layer (SSL).

- The security services are accessed using commands such as TPM_Seal, TPM_Bind, TPM_Sign, and others. Upon its reception, the command is verified, decoded, and executed with the aid of the cryptographic engines as far as needed. The commands are closely related to the underlying architecture. Therefore, they are settled on the TPM Command Layer (TCL), which directly precedes the TPM Hardware Layer (THL).

- The THL represents the TPM architecture itself, which includes all the cryptographic engines and corresponding protocols. Note that THL, in this context, is simplified and should refer to the underlying hardware/software architecture.

Thus, a compromise in one of the cryptographic engines of the TPM would affect all the layers in the layered abstraction

Threat	**Affected Components and Functions**	**Countermeasure**
Side-Channel attacks on RSA	RSA, keygen., Volatile and Non-Volatile Memory	Side-Channel aware RSA Implementation
Weak RSA Implementation	RSA, keygen., Volatile and Non-Volatile Memory	RSA with longer key
Broken RSA	RSA, keygen., Volatile and Non-Volatile Memory	Non RSA solution
Broken RSA Key Generator	EK, SRK, Key Hierarchy	New Key Generator
Broken SHA-1	PCRs, Attestation, TPM_SHA1 commands	New Hash function
Side-Channel attacks on HMAC	Secret Key, User_Auth_Data, Integrity	Side-Channel aware HMAC Implementation
Broken HMAC	User_Auth_Data, Integrity	New HMAC
Side-Channel attacks on RNG	Random Numbers, keygen., Nonces	New RNG
Insecure Communication Interface (LPC)	Keys, PCRs	Secure Communication Interface & Protocols
Broken Firmware (ROM)	All TPM_Commands	New Firmware

Table 2.4: Threats and Necessary Countermeasures[64]

model. In this regard, Table 2.4 gives an overview of the individual threats a TPM has to deal with, the affected components, and the countermeasures to the posed threats [64]. With reference to the Table 2.4 and Table 2.1, it is clear that a compromised cryptographic engine would affect the security functions provided by the TPM. Thus, there is a necessity to address the issue of broken/compromised/weakened cryptographic engines on the TPM. For this, a novel concept, referred to as STC, is proposed in this thesis. Utilizing this concept, a novel architecture,

referred to as STPM, which provides a flexible and secure update of the cryptographic engines on it when they are compromised, is designed. The concept of STC and the corresponding STPM architecture are illustrated in the next chapter.

3

SUSTAINABLE TRUSTED COMPUTING

The main goal of this thesis is to provide a novel architecture, i.e., the STPM for the user, as depicted in Figure 3.1 (high level view), which supports a flexible and secure update of cryptographic engines on it. This depicted architecture, in contrast to the conventional TPM architecture depicted in Figure 1.4, comprises of updatable cryptographic engines i.e., the *Key Generator*, RSA, SHA-1, HMAC, and RNG may be replaced with new uncompromised engines in case of their compromise [39]. With such an architecture, the user is provided with an uninterrupted availability of the TPM security functionalities even if one of the cryptographic engines on it, is compromised. This flexibility is achieved by an on-the-fly update of the compromised engine with a new uncompromised engine. Further, the user can load the STPM with side-channel resistant or quantum computer resistant cryptographic engines to avoid the related attacks and thus make the device more secure.

This chapter explains about the requirements and specifications for designing the novel STPM architecture followed by an illustration of the proposed architecture itself. Additionally, a description of the update procedure is given, utilizing which the compromised cryptographic engines on the proposed STPM architecture are updated with new ones. Furthermore, an illustration about the significant task of re-establishing the trust in the system after an engine update is provided at the end.

The need for the STPM may arise because of one or more of the following reasons:

- A compromised cryptographic engine on a TPM leads to a compromise in security functionalities provided by it. One solution could be to turn-off the TPM completely but this may not be desired by the user.

- There may be implementation shortcomings crept in the TPM during its manufacture, which cannot be changed any more due to its current design (Application Specific Integrated Circuit (ASIC)), after its manufacture and deployment.

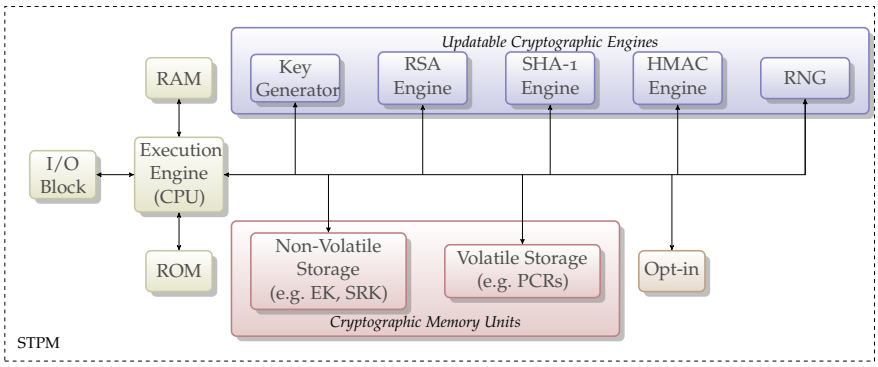

Figure 3.1: Proposed STPM Architecture

- Replacing a compromised TPM every time with a new one is not economically feasible considering the scale of its deployment in the market.
- Though hard-wiring of cryptographic engines into the TPM is regarded as a security feature, so far, this approach is useless with the new developments in cryptanalysis techniques and emergence of novel powerful computer systems such as quantum computers.

Considering that the current trust anchors are unable to address these new challenges and does not guarantee security against attacks, there is a need for designing a new TPM architecture. However, before delving into the architectural and functional details of the STPM, we first describe the update procedure at an abstract level. The update procedure of the STPM as a whole represents a distributed system comprising of an update server (a computer) with new cryptographic algorithms, a host computer (PC) with an embedded STPM, communication between the update server and the host, and the internal communication between the components of the STPM, as depicted in Figure 3.2. To perform an update and to provide the TPM functionalities, the STPM holds an update algorithm component and a component for executing the TPM commands respectively. Further, it consists of the locations themselves for loading the new cryptographic engines by the update algorithm.

The server communicates with the PC over an external communication channel (e.g., over the Internet) as in a conventional server-client system. This communication is secured utilizing a Public Key Cryptography (PKC) mechanism for authenticity verification of the entities involved and the integrity verifica-

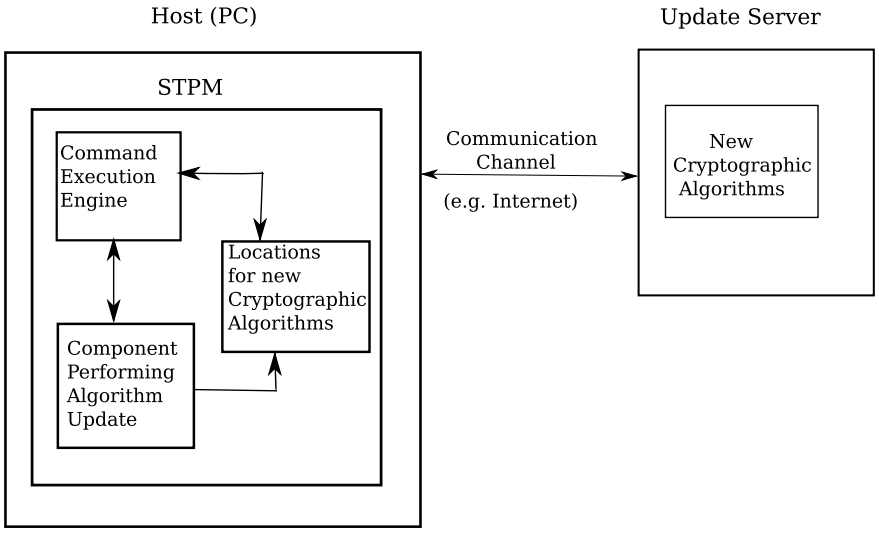

Figure 3.2: System Level View of the STPM Update Procedure

tion of data being exchanged. The received update data by the host is converted into an update command by the processor of the host PC and is then transferred to the STPM which further processes it utilizing its internal components before configuring the designated locations on it. Later an acknowledgement about the status of the update is sent back to the update server by the host. However, a detailed discussion about all these steps is presented later in the chapter.

3.1 FUNDAMENTAL ARCHITECTURAL REQUIREMENTS FOR STPM DESIGN

To simultaneously support an update of the cryptographic algorithm and to execute the commands, the STPM architecture must contain both static (non-updatable) and reconfigurable (updatable) regions on it. The static region contains an execution engine which has to run un-interruptedly for executing the TPM commands and the reconfigurable region holds the locations for updatable cryptographic engines of the STPM, to be loaded by the update algorithm. This update algorithm must also be located in the static region of the STPM. In addition to requiring the updatability feature, the STPM architecture must fulfill memory (storage), interface (bus), and other requirements as detailed below.

With reference to the conventional TPM architecture (c.f. Figure 1.4), it is clear that a Non-Volatile Memory (NVM) is required to store sensitive cryptographic data i.e., EK, SRK, and the platform certificates persistently. However, such a memory is not inherently available with STPM, thus it must contain an external NVM to store this data. Further, the NVM in our case should be programmable to support the loading of new keys in case of an asymmetric engine update i.e., RSA by Elliptic Curve Cryptography (ECC), which mandates the replacement of old platform keys in case of compromise. There exist such memories (e.g., Erasable Programmable Read Only Memory (EPROM), Electrically Erasable Programmable Read Only Memory (EEPROM), and Flash) already in the market, which may be utilized. Additionally, this NVM has to be large enough to hold the keys for cryptographic operations to be performed by the update algorithm of the STPM. These keys together with the STPM cryptographic keys are programmed into this NVM by the STPM manufacturer during its manufacture. Also, a re-writable memory is needed to store the hash values representing the system state and the bitfiles for the static/dynamic regions, which may be accomplished by utilizing a programmable Flash. A bus (such as Serial Peripheral Interface (SPI)) external to the microcontroller is required for accessing the content inside the NVM and the Flash. These memories are attached to this bus as peripheral components and are interfaced through corresponding memory controllers. Furthermore, an interface (bus communication) between the PC and the STPM itself is required, as done by LPC bus. Additionally, as a security requirement the communication between the NVM and the STPM must be protected against an unauthorized access, to avoid the compromise of the keys inside it and there by loss of all the data secured by them.

3.2 STATE OF THE ART

An investigation has been done, as illustrated below, to find out the availability of suitable platform in the market which supports these fundamental requirements of the STPM.

Though structured-ASICs with both static and reconfigurable parts seem to be a solution, they are only mask programmable, an operation which can only be performed by the chip manufacturer [100]. With regard to the above statement, an FPGA supporting the PR feature (c.f. 4.2) seems to be a well-suited platform for the deployment of updatable TPMs. However, major-

ity of the current Static Random Access Memory (SRAM) based FPGAs do not support an on-board NVM yet, as required by the TPM for storing sensitive cryptographic data such as EK, SRK, and other certificates. Some FPGAs, such as the *Spartan-3AN* family from *Xilinx*, contain an on-chip non-volatile Flash memory to keep the configuration data [106]. Such FPGAs, however, belong to the low-price class and do not support PR, a feature required for providing static and reconfigurable parts simultaneously in the FPGA.

The high-end Virtex-6 FPGA from *Xilinx* has an on-chip bitstream HMAC algorithm implemented in hardware [29]. When used, this provides both the integrity and the authenticity verification of the incoming bitstream before loading it onto the device. However, these devices are volatile in nature and do not provide the required NVM for the design of STPM. The other possible platform may be the *Nextreme* from *eASIC*, a family of new ASIC devices, which combines an FPGA like logic cell, called an *eCell*, with single customizable via routing [35]. In contrast, a hybrid ASIC/FPGA was proposed jointly by *IBM* and *Xilinx* [110]. This design is an incorporation of the FPGA cores into the ASIC which allows the programmable circuitry to enable a single physical chip design to satisfy several different applications. These architectures are also not suitable for the STPM design as they too lack one or the other aforementioned required features.

Meanwhile, there exist some Flash-based FPGAs, such as *SoC FPGAs* from *Microsemi*. The advantage with the Flash-based FPGAs is that the configuration data is not lost even after power-off of the device. Furthermore, the latest device from this series, referred to as *SmartFusion2*, is capable of providing robust design security [4]. This device supports the features such as a non-deterministic RNG, a set of cryptographic algorithms (i.e., ECC, Secure Hash Algorithm-2 (SHA-2), AES, and HMAC) and an SRAM based Physical Unclonable Function (PUF) in addition to others. Utilizing these, an architecture may be designed that is secure against threats such as cloning, counterfeiting, and tampering. Further, it provides security functionalities such as integrity and authenticity verification, which are required for a secure update of cryptographic algorithms on the STPM.

However, the Flash-based FPGAs such as the *SmartFusion2* are not in common use yet, in contrast to the SRAM based FPGAs. This is because the latter have high logic density resulting in higher capacities and higher performance when compared to

the former. Furthermore, the Flash-based FPGAs require additional processing steps above a basic Complementary Metal Oxide Semiconductor (CMOS) process and thus resulting in a lag in the CMOS process. In specific, the *Microsemi* Flash-based FPGAs are manufactured at 65 nm while the *Xilinx* SRAM based FPGAs at 40 nm thus the latter technology yielding in a larger and cheaper devices. Additionally, the integration of additional features as done in *SmartFusion2*, at the end leads to a very complex design. For these reasons, an SRAM based FPGA supporting the PR feature is utilized to design the STPM in this work.

In the following we detail some existing approaches in the literature to build trusted platforms based on an FPGA.

The rise in the usage of FPGAs in building embedded systems has led to a mandatory requirement to protect the IP in such systems. In this context, the authors, Eisenbarth et al. in [36], have tried to extend the FPGAs with the TC functionalities so that a reconfigurable application is bound to the underlying TPM and even to bind any higher layer software to the whole architecture. For this, the TPM, stored as a bitstream in the external memory, is loaded onto the FPGA at start-up and subsequently store its cryptographic measurement in a register for later use. Consequently all the loaded software applications on the FPGA are integrity measured by the activated TPM before their execution on the device. The loading of the TPM as a bitstream provides the flexibility and scalability in update of the TPM functionalities, in particular the hardware-based cryptographic engines and the accelerators. However, the proposed work only points out the feasible implementation options and associated challenges but does not provide a concrete implementation of the proposed architecture yet. Further, the authors do not discuss about the measures to be taken, after an update of TPM functionality, in order to re-establish the trust in the system.

Further, Glas et al. have proposed a method for mapping the TPM specification to the embedded systems [46]. They built a trusted embedded platform based on an FPGA that additionally supports the dynamic partial reconfiguration feature. For this purpose, they attached an external block, with an off-the-shelf TPM, to the system and made few modifications to the interface between the system and this external block. Thus, the main goal of this work is to provide a secure dynamic partial reconfiguration of the FPGA utilizing the TC features. In another work, Ekberg et al. have proposed a mobile system that utilizes the trusted computing concepts, referred to as Mobile Trusted

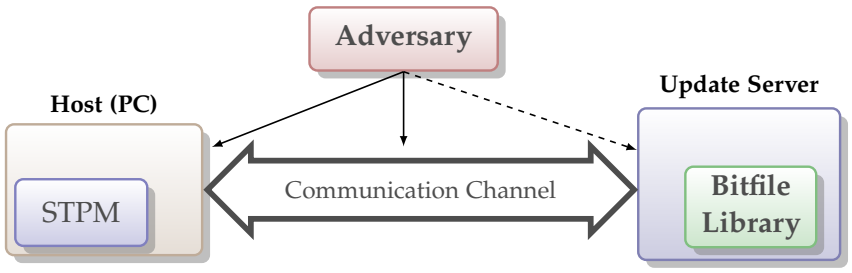

Figure 3.3: Update Environment

Module (MTM), as detailed in [37]. In here, the task of MTM is to provide TPM like features to the multiple stake holders, i.e., application provider, user, and network provider, in a mobile computing system. A new architecture for the TPM that utilizes a reduced trust boundary is proposed by Kursawe et al. in [60]. This architecture is much smaller and simpler implementation without sacrificing security gains from a classical TPM design. However, to the best of our knowledge, none of the above approaches addresses the issue of updating a compromised cryptographic engine on the TPM and re-establishing the trust in the system after its update.

3.3 ADVERSARIAL MODEL

As mentioned before the goal of the STPM architecture is not just to achieve an update of the cryptographic engine by the update algorithm but also to perform this update securely. Thus, we consider an adversarial model for the update procedure and evaluate the security of the STPM against an adversary. During the STPM update, two parties are involved: the STPM residing in a computer system and the update server maintaining the bitfile library for new cryptographic engines, as depicted in Figure 3.3. The computer system (PC) embeds the STPM onto its motherboard as in the case with a conventional TPM. This PC is assumed to be part of an enterprise environment and is administered by the dedicated staff. It is assumed that the STPM itself trusts the PC else additional security concerns will arise, which are out of scope here. The update server delivers the new cryptographic engines to be loaded onto the STPM over an insecure network. Thus, the **API!** (API!) corresponding to the network must be updated to provide a secure transfer of the data. Further, it is administered by the STPM manufacturer, who initiates

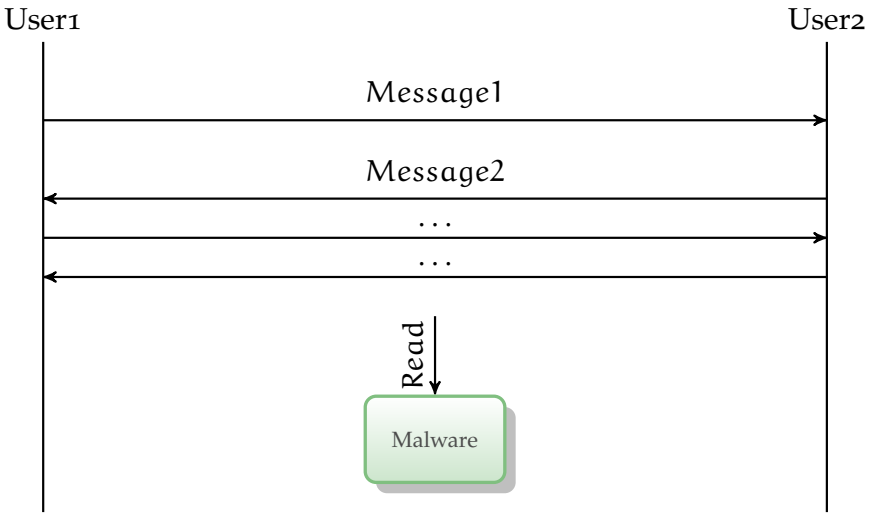

Figure 3.4: Cloning by Interception

and provides all the update services. Additionally, the update server is assumed to be run in a secure environment i.e., attacks on the server itself, such as denial-of-service attack, are not considered in this thesis.

A successful attack on a cryptographic engine on the STPM is detected only by the update server. This scenario is similar to a virus attack on the user system, in which an user reacts only after he comes to know about the attack. With reference to the taxonomy of attacks given by Popp [75], some of the possible attack scenarios, during the update of STPM, are explained in the following.

The adversary in the considered model is an active eavesdropper i.e., someone who first taps the communication line to obtain messages and then tries everything in order to discover the plain text [34]. More precisely, the adversary may obtain any message passing through the channel, an attack referred to as cloning by interception. This attack may lead to copying of the bitfile data being transferred from the update server. An example case of cloning is depicted in Figure 3.4. In here, two users (User1 and User2) are exchanging messages over a communication channel, which is intercepted by an unauthorized observer i.e., the malware, to read the messages and store them for later use [53].

The adversary can perform replay attacks i.e., send false material utilizing the data he obtained during the interception of the channel. Replay attacks are those in which the intercepted

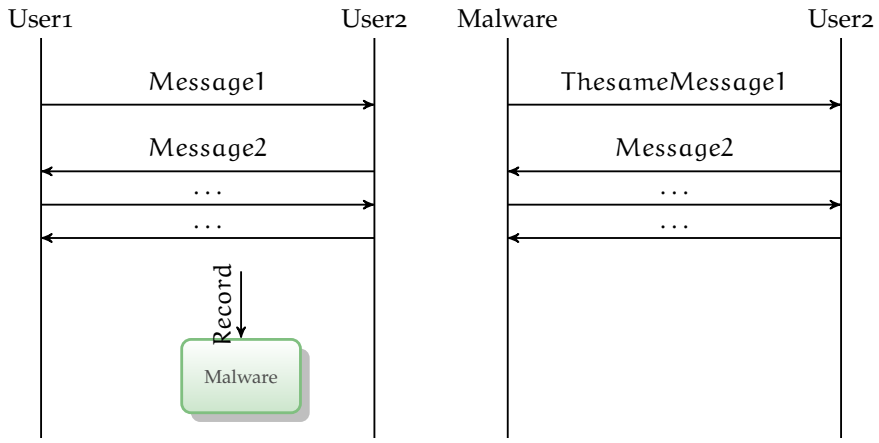

Figure 3.5: Replay Attack

data or credentials from the previous session are retransmitted by the attacker to fraudulently authenticate to one of the users. Such a scenario is depicted in Figure 3.5. The malware records the previous transmissions between user1 and user2 to authenticate itself to the user2 later. In our case, User1 and User2 may refer to STPM and update server respectively.

The other kind of possible attack is a malicious update of the STPM by a man-in-the-middle attack (c.f.. Figure 3.6). Here in, the attacker deceives the STPM to authenticate himself to be the update server or deceive the update server to authenticate himself to be the STPM. In the former case, the attacker may load malicious configuration data onto the STPM to destroy its functionality and in the latter he would know the content of the update data. The aforementioned attacks are categorized as classical cryptanalysis attacks, which can be avoided by the proposed update algorithm as detailed later in this thesis.

There exist another group of attacks referred to as implementation attacks such as side-channel attacks, reverse engineering, and others. The side-channel attacks make use of the power consumption measurements of the device, while performing certain functionality, to extract the secret data. In reverse engineering, the attacker tries to reconstruct the functional description in the form of netlists from the initial configuration data. In case of a physical attack, the adversary physically possesses the device and may destroy it to extract secrets from it.

However, the attacker in the considered model here is assumed to be able to perform only classical cryptanalytic attacks and not the implementation attacks. Furthermore, this assump-

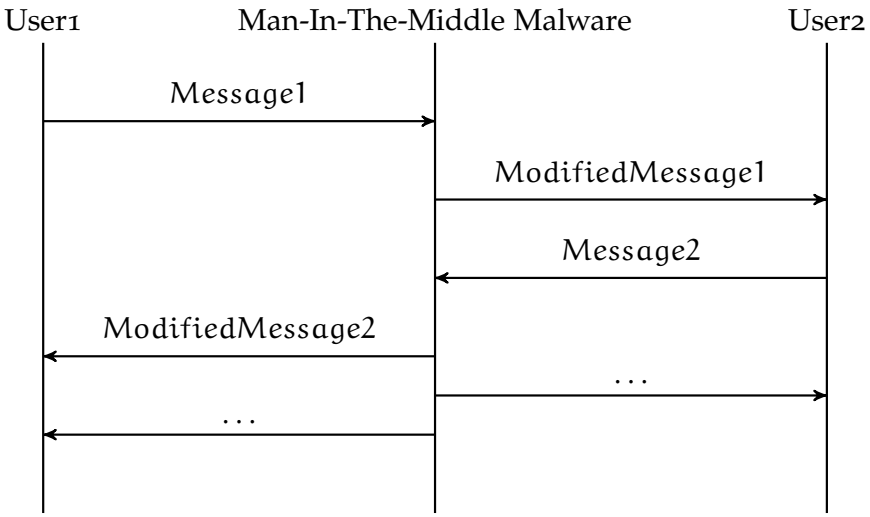

Figure 3.6: Man-In-The-Middle Attack

tion complies with the TCG specification [89], which states that the physical and the implementation attacks on the TPM during shipping and after deployment are not considered.

3.4 STPM ARCHITECTURAL SPECIFICATIONS

This section deals with determining necessary architectural specifications for the STPM design in addition to the aforementioned fundamental design requirements in Section 3.1. These specifications in turn are dependent on the functional and non-functional requirements arising when replacing the compromised cryptographic algorithms on the STPM and later re-establishing the trust in the system. For this, two fundamental cryptographic algorithms on the STPM i.e., SHA-1 and RSA are considered. The functional and non-functional requirements of the updates may be derived based on the resulting implications when these algorithms are compromised, as explained in the sequel.

3.4.1 *Compromised SHA-1*

Compromised SHA-1 has the following implications on the system (denoted as I):

- I1: The computed platform state, i.e., the integrity measurements stored inside the PCRs, are no more valid.

- I2: The signatures generated (utilizing SHA-1) during remote attestation may be forged.
- I3: The data sealed to the platform can no more be unsealed.
- I4: Command authorization by HMAC is incorrect because it utilizes the same SHA-1 algorithm.

Based on these implications, following functional (denoted as FR) and non-functional requirements (denoted as NFR) arise:

- FR1: The update of SHA-1 with a new hash algorithm, for example SHA-2. Thus, the update of the compromised algorithm by a new one itself is the first and foremost functional requirement.
- FR2: To handle I1, a new platform state has to be computed with new hash algorithm.
- FR3: To handle I2, the signatures have to be recomputed with new hash algorithm.
- FR4: To handle I3, unsealing the sealed data with old hash engine and then sealing it to the new hash engine is required.
- FR5: To handle I4, the HMAC should be configured to make use of the new hash engine.

3.4.2 *Compromised RSA*

Compromised RSA has the following implications:

- I1: The complete key hierarchy of the TPM is compromised.
- I2: The signatures generated (utilizing RSA) during remote attestation may be forged.
- I3: The bound and sealed data is lost because the keys utilized for these operations are the RSA generated asymmetric keys.
- I4: The cryptographic engines of the TPM such as the key generator, asymmetric encryption/decryption engine that rely on RSA are no more trustworthy.

Based on these implications, the following functional requirements arise:

- FR1: The update of RSA with a new asymmetric algorithm such as ECC.
- FR2: To handle I1, a completely new TPM key hierarchy is required to be built.
- FR3: To handle I2, the signatures have to be recomputed with new asymmetric algorithm.
- FR4: The bound/sealed data is already lost and can not be recovered, thus only the new data can be bound/sealed with a new asymmetric engine.
- FR5: To handle I4, the firmware should be updated to execute the related commands utilizing the new asymmetric engine.

However, the non-functional requirements are common for both SHA-1 and RSA update and may be generalized as follows:

- NFR1: The update procedure should be secure.
- NFR2: The update should be performed very rapidly.
- NFR3: A de-allocation of the resources that are allocated to the previous engine should be done.
- NFR4: Over-provisioning of the resources for the new engine must be handled.
- NFR5: The overall STPM architecture design must be economical (i.e., cost-aware).

3.5 GENERIC STPM ARCHITECTURE

Based on the aforementioned fundamental architectural requirements, functional, and non-functional requirements, a generic STPM architecture as depicted in Figure 3.7 is proposed, which resembles the chip design suggested in [15].

The static region contains an execution engine (a microcontroller) for executing the TCG specific commands given to the STPM by the CPU of the system. These commands, which arrive over the LPC bus (as in current TPM standard) are received by the execution engine over its I/O block [51]. These commands

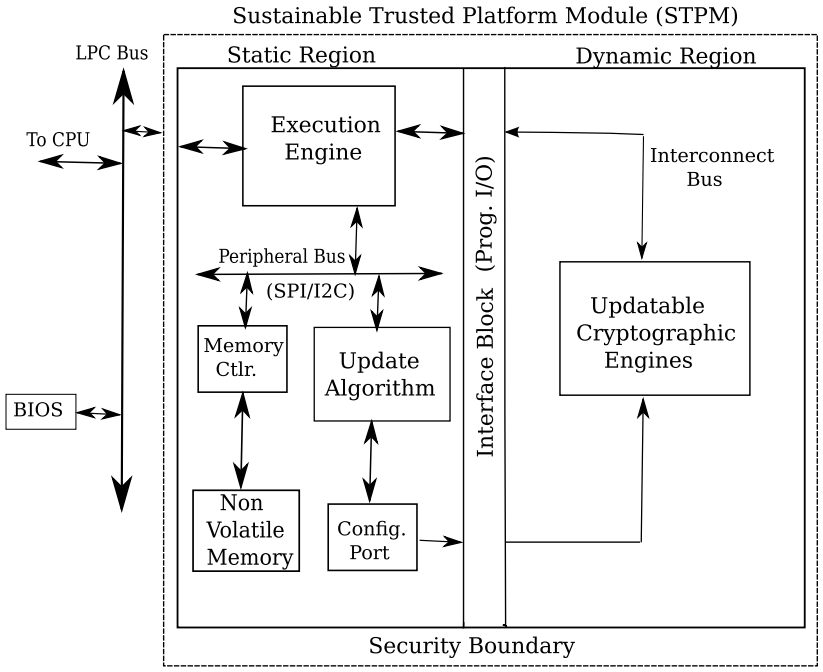

Figure 3.7: Generic Architecture of STPM

are then executed by the processor of the execution engine to provide the required security functionalities such as integrity measurement, remote attestation, binding, and sealing.

In contrast, the dynamic region holds locations for the updatable cryptographic engines of the STPM, to be loaded by the update algorithm through the configuration port. The update algorithm and the configuration port are both in the static region. The new cryptographic engines arrive as an update data from an update server to the CPU of the system over an external communication channel and through to the execution engine of STPM over the LPC bus. Before loading the new engines, they are authenticity verified and decrypted by the update algorithm.

Additionally, the static region contains an NVM for storing the sensitive cryptographic keys and data. This NVM is accessed by both execution engine and update algorithm through a memory controller (Memory Ctlr.) interface. In specific, the NVM stores the cryptographic keys required for the operations to be performed by these two components. Further, it can be seen that the BIOS is also attached to the same LPC bus to which the STPM is connected, thus complying to the standard TCG specified TPM design.

Input Message Block

Authorization Tag	Parameter Size	Command Code	Pay Load

Output Message Block

Authorization Tag	Parameter Size	Return Code

Figure 3.8: Input and Output Message Blocks of Update Command

In the following, a detailed description is given about how the generic STPM architecture meets the aforementioned specifications. While doing so, further working details of the individual components of both static and dynamic region are also given.

To update the compromised algorithm with a new algorithm, the STPM architecture should be equipped with an update algorithm. Further, this functional requirement of having an update algorithm is common for both SHA-1 and RSA engine update. Thus, to fulfill the requirement of an algorithm update, one needs update data, an update command, and then the processing of this update command by the execution engine, which in turn utilizes the update algorithm.

The update data comprises of an encrypted new cryptographic engine and an attached authorization data (c.f. Section 3.5.2). The update command comprises of message blocks (input and output) similar to a conventional TPM command (e.g., *TPM_OIAP* command) as depicted in Figure 3.8.

The input message block contains the authorization tag, the parameter size, the command code, and the payload. The authorization tag (2 bytes) indicates whether the command has to be authorized or not before its execution. A command is said to be authorized when it provides the 160-bit long hash representation of the owner password, referred to as *Auth_Secret*, before its execution by the TPM. The owner password is generated while taking ownership of the TPM, which is encrypted by the public part of EK and is stored inside the NVM of the TPM.

The parameter size (4 bytes) specifies the total number of bytes that make up the command and the command code (4 bytes) specifies the command to which it is actually referring to. The payload (variable in size), depending on the command

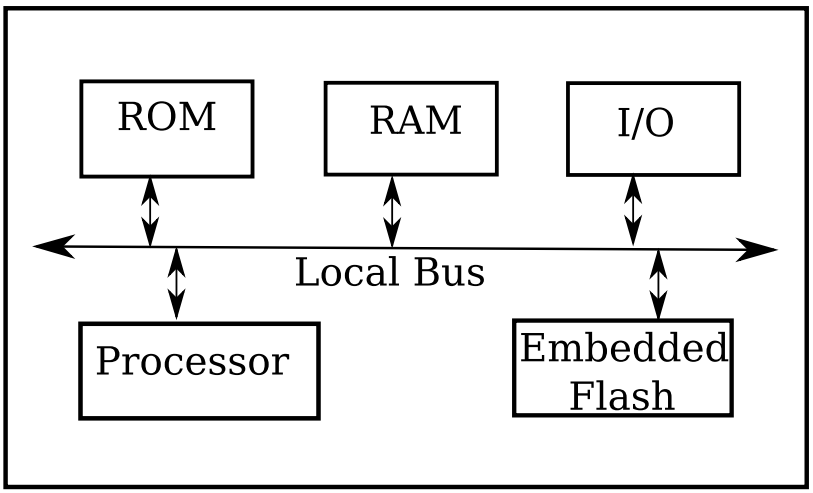

Figure 3.9: Execution Engine of STPM

type may refer to one or more parameters of various lengths such as handles, nonces, hashes, and data. However, in our case the payload is the aforementioned update data. The output message block is identical to input message block except that the return code is now a status signal indicating the successful or failed execution of the command.

Once the update command with update data is available at the STPM's execution engine, the command is executed by its processor, and the update data is fed to the update algorithm. The internal details of the execution engine and the update algorithm are described in sequel.

3.5.1 *Execution Engine*

One significant component in the static region of the STPM is the execution engine which provides the required control flow and the corresponding command execution similar to a conventional TPM. In its design, the execution engine of the STPM is a microcontroller system, which comprises of several internal components such as a processor, RAM, ROM, I/O ports, and an internal local bus connecting all these components as depicted in Figure 3.9. Though the most commonly used microcontrollers are the 8-bit, there also exist 16 and 32 bit microcontrollers in the market and the user may select the one based on his specific requirements. The bit width (8, 16, and 32) here indicates the data width, which the microcontroller, in specific its

processor, can perform operations on, in a single clock cycle. In our case, we also utilize an 8-bit microcontroller inside the STPM because it has to interface with the LPC bus for command and data transfer. With regard to this, the I/O read and I/O write cycles of the LPC bus also support a data size of 1 byte (8-bits), which again complies to the standard TCG specifications.

Further, in case of STPM design, the "update command" and corresponding "program code" should also be included in the standard command set and the ROM respectively. For the command execution, the processor utilizes the program code (i.e., TCG firmware) stored inside the ROM and the working memory RAM, which are attached to the processor local bus, and are accessed through the provided address and data lines to the processor. Further, the integrated ROM inside the microcontroller should be programmable (e.g., EPROM or EEPROM) because it needs to be updated to support new TPM commands in case of an engine update or for adding additional features to the STPM.

The I/O block in the execution engine is a set of programmable input/output ports. These ports can be thought of as memory cells or registers that are connected to the processor through the data bus and further to the outside world (to the CPU of the system and other peripherals) via pins on the side of the microcontroller. For example, the data from the CPU, such as the STPM command or other data that propagate over the LPC bus are fed to the STPM, in specific to the I/O block, which in turn is fed to the processor of the microcontroller through its data bus for further processing.

There exists an additional essential component inside the execution engine i.e., the embedded Flash memory. This memory is required to store the configuration data, for loading into the dynamic region of the STPM, because when the STPM is switched-off, the dynamic region loses its content because of its volatile nature. Further, this component stores the STPM computed platform state (i.e., the PCR values). The processor accesses this embedded Flash memory through a latch buffer comprising of two latches and a transceiver. A microcontroller with such a memory design is already available in the literature [93].

3.5.2 *Update Algorithm*

The update algorithm in our case is a hardware module, as depicted in Figure 3.10, with the cryptographic blocks for encryption, decryption, and authentication operations. Further, the up-

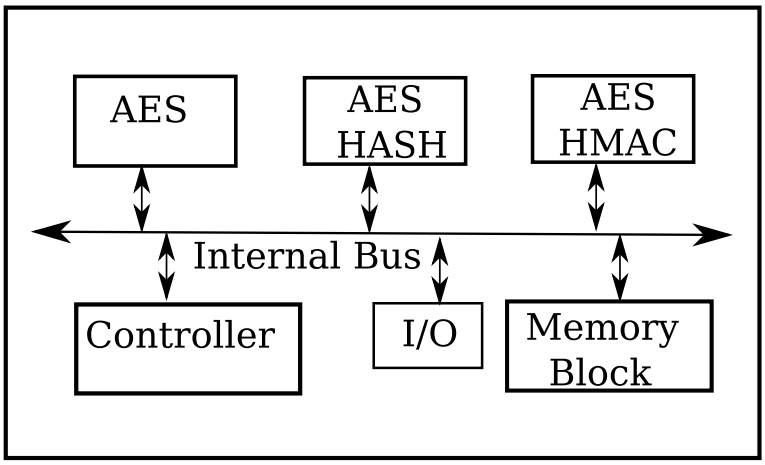

Figure 3.10: Internal Components of the Update Algorithm

date procedure should be simple in design flow and must comply to the TCG standard specifications in terms of security it provides, which is achieved by the proposed algorithm as detailed in the sequel. Whereas the encryption and decryption operations utilize the AES block, the authentication utilizes the AES-Hash and AES-HMAC (referred to as Cipher-based Message Authentication Code (CMAC)) blocks. Thus, such a hardware module not only is easy to design but also consumes few computational resources of the underlying platform because it reutilizes the same base block, i.e., AES, for all its operations.

The hardware module additionally contains a controller (a state machine), for performing the aforementioned cryptographic operations, and a memory block acting as a data buffer for temporary storage of incoming and outgoing data. The hardware module, in specific its controller, accesses the configuration port in the static logic through its I/O block. This configuration port is utilized for loading the new cryptographic engines into the dynamic region of the STPM.

The update algorithm is embedded in the static region of the STPM and communicates with the STPM processor over the SPI bus (Figure 3.7). The SPI is an 8-bit or more synchronous serial interface providing a master-slave relationship [78]. In our case, the master being the STPM processor and the update algorithm being the slave. This update algorithm is also used in another work for performing a secure and an authenticated update of the device IP in an FPGA-based embedded system [40].

As mentioned earlier, the update algorithm must assure both secure and timely loading of the new cryptographic engines on to the STPM from the update server, thus satisfying the first and second non-functional requirements. To achieve this, the update process should be secure against attackers and should be performed in a short time. It is clear that an authenticated communication between the update server and the host guarantees a secure transfer of update data. For this purpose, a special data format is specified for the update data, denoted as U_{DATA}, originating from the server.

Following is a notation introduced before defining the update data:

$config$	The configuration file for the new engine
k	Previously exchanged symmetric encryption key for IP protection
$E(k, M)$	Symmetric encryption of message M with key k
k_{cmac}	Key used for generating the CMAC
$CMAC(k_{cmac}, M)$	CMAC of message M computed with key k_{cmac}

Using this notation, the update data is defined as:

$$U_{DATA} \Leftrightarrow (E(k, config)||CMAC((k_{cmac}, E(k, config)))) \quad (3.1)$$

From the above definition, it can be seen that the update data consists of encrypted configuration for the new engine and the element for verifying its authenticity (i.e., an attached CMAC [69] of the encrypted data). The CMAC algorithm operates identical to an HMAC algorithm, except for the utilized MAC to generate the output. The generated MAC in the latter is based on a conventional hash algorithm (e.g., SHA-1) whereas in the former it is based on a block cipher based hash algorithm (e.g., AES-Hash).

Upon receiving the update data at host, it is first converted into an update command and is then transferred to the STPM, in specific to its execution engine. Then the execution engine interprets the update command and sends the payload (i.e., the update data) part of it to the update algorithm over the SPI bus for authentication and decryption operations. To perform these operations, the received data is first stored in the temporary

Algorithm 1 Cryptographic Engine Update Algorithm

Require: U_{DATA}, symmetric key k, cmac key k_{cmac}

1: Load U_{DATA} from server into the host
2: Send the update command to the STPM
3: Compute $CMAC((k_{cmac}, E(k, config)))$ and compare with attached CMAC
4: **if** Comparison result is true **then**
5: Decrypt $E(k, config)$
6: Configure STPM with $config$
7: Activate new engine
8: **return** Update_complete.
9: **else**
10: Reject data and send failed command
11: **return** Update_failed.
12: **end if**

memory (i.e., memory block of the update algorithm) and is fed to the authentication block (AES-HMAC) in 128-bit blocks. This memory is large enough to hold the complete payload and to store the intermediately computed CMAC for later comparison. Once the complete CMAC on the encrypted data is available, it is compared with attached CMAC in the payload. If the values match, it implies that the authentication is successful i.e., the update data comes from an authentic server, and thus the encrypted configuration data can further be decrypted. If the values do not match, it implies that the authentication has failed, thus the data should be rejected and the update process should be stopped by aborting the update command and returning a failed command code.

Further actions that need to be taken based on the authentication result of the controller are as detailed below:

- Case 1: Authentication successful
  The decrypted data is sent back to the STPM processor over the same SPI interface for storing it in the embedded Flash which is later required for configuring dynamic region of the STPM on next start-up of the system. After storing a copy of the configuration data in the Flash memory, it is further programmed into the specific location in the dynamic region by the controller of the update algorithm through the configuration port. The status of successful update command is acknowledged to the server via the host system.

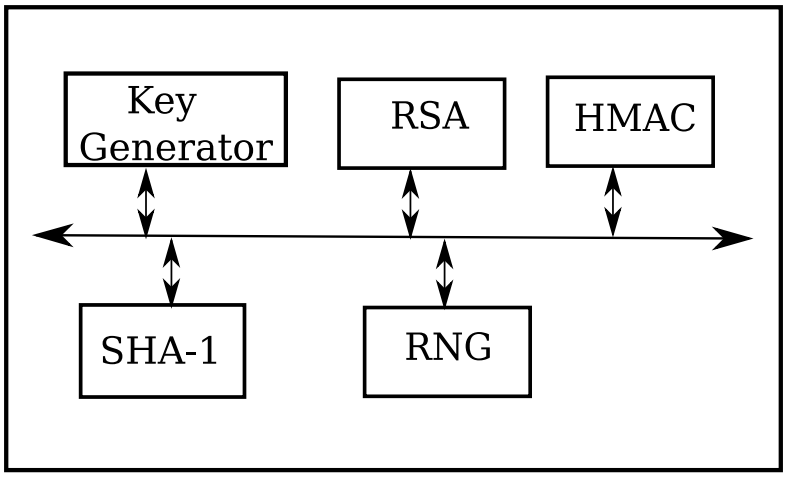

Figure 3.11: Updatable Cryptographic Engines of the STPM

- Case2: Authentication fails
  The rejected data is sent back to the STPM processor, which in turn deletes the data from the system by standard data deletion mechanisms. The status of failed update command is acknowledged to the server via the host system.

The procedure for loading a new cryptographic engine on to the STPM is detailed in Algorithm 1. It can be seen from the algorithm description that, for verifying the attached CMAC on the update data, the STPM requires the storage of the CMAC key, k_{cmac}, in the NVM within the STPM. Further, to decrypt the encrypted cryptographic engine, the STPM must be equipped both with the symmetric cryptographic engine (AES) and the corresponding symmetric key k. These keys are programmed into the NVM of the STPM during its manufacture as mentioned in Section 3.1.

3.5.3 *Updatable Cryptographic Engines*

The dynamic region of the STPM comprises of locations for updatable cryptographic engines as depicted in Figure 3.11. These locations are reconfigurable to support an engine update in case of a compromise.

These cryptographic engines are accessed by the processor of the execution engine through the interface block (programmable I/O and an interconnect bus) of the STPM architecture for executing the various TPM commands. The processor of the micro-

controller treats the dynamic logic as a large block of memory (as in a Memory Mapped Input Output (MMIO) scheme) with specific locations performing certain cryptographic operations and accesses them over an interconnect bus as depicted in Figure 3.7. Though only a compromise of SHA-1, RSA, and their dependents have been discussed till now, it can be seen that RNG is also a part of updatable cryptographic engines block.

To summarize, the correct functioning of the STPM architecture relies on a coordinated operation of all its three essential blocks (i.e., execution engine, update algorithm, and updatable cryptographic engines) and their inter connection and communication (i.e., through buses and interfaces) along with the other essential components (i.e., memories and keys). Further, all these components of the STPM architecture are protected by a secure package, referred to as security boundary in Figure 3.7, to avoid any physical and implementation attacks on them.

3.6 STPM ARCHITECTURE ANALYSIS

Given the STPM architecture, the update server, and the defined update algorithm, this section analyzes how the requirements that raised due to compromised SHA-1 and RSA (as detailed in Section 3.4.1 and Section 3.4.2 respectively) are met by the proposed STPM architecture. Meeting these requirements is to regain or re-establish the trust in the system because the TPM is considered as the trust anchor of the system and thus a compromised engine on it implies a compromise of trust kept in the system utilizing it.

The first functional requirement of the STPM architecture is to perform the update of compromised SHA-1 and RSA engines with correspondingly new uncompromised engines. This is achieved by utilizing the aforementioned update algorithm as detailed in Section 3.5.2. The other individual (SHA-1 and RSA) functional requirements to be met by the proposed STPM architecture are detailed in the following.

3.6.1 *Regaining Trust after SHA-1 Engine Update*

3.6.1.1 *Integrity Measurement*

Integrity measurement is a basic function to compute the hash values to be stored in PCRs, which represent the current system state. In the case of an hash engine update, this representation

of the system state is no more valid, because it had been computed with the old hash engine. This leads to an undefined representation of the system state, where it cannot be determined whether the system is in a trustworthy state or not. Even if there are no changes in hardware or software, the current PCR contents will differ from future values computed with the updated hash engine. Also, due to the fact that the measurement has been performed using the old hash engine, the PCR contents are prone to collision attacks. However, after updating the hash engine, the whole platform configuration is recomputed and stored in the PCRs. Because most of the PCRs are not resettable, the system has to be rebooted before the new measurements can be considered as trustworthy. To indicate the trustworthiness, the system state has again to be compared to the reference values.

Further, the PCRs should now support the storage of hash values generated by new hash engine, which causes their size to be changed. This is because the hash value computed by the new engine, for example SHA-2, may be 224/256/384/512 bit unlike the 160 bit hash generated by SHA-1. This implies that the memory storing the PCR values should be flexible in size to accommodate larger values. For this, the embedded Flash memory included inside the execution engine during the design of the STPM should be large enough.

3.6.1.2 *Remote Attestation*

A trusted platform is able to attest the current system state to any requester. Remote attestation is the only function, which is able to directly distinguish between trustworthy and untrustworthy system states. Usually, the trusted system states are stored in RMLs generated by the IT-department of the organisation. In the case of an update of the hash engine, these RMLs become invalid. Therefore, the reference list has to be recomputed by means of the updated hash engine. Thus, the contents of the PCRs have to be re-evaluated in order to reflect the update in the stored hash values with the new hash engine.

3.6.1.3 *Binding*

Binding is used to asymmetrically encrypt sensitive data with keys, which are available on a specific TPM only. The binding function can be considered to be independent of the trust someone puts into the system. Of course, the confidential data has

to be protected properly for which the binding procedure itself does not provide any measures to reflect the trustworthiness of the current state. Therefore, the update of the hash engine does not affect the binding function directly.

3.6.1.4 *Sealing*

Sealing uses the current system state to protect sensitive data. It extends the binding operation by including the system state as one of the parameter along with the data and the keys. Sealed data can only be unsealed if the system is in the same state as during the sealing procedure. After an update of the hash engine the data cannot be unsealed, because the PCR content has changed. To tackle this problem, a resealing process must be performed as detailed below.

3.6.1.5 *Data Resealing*

Data resealing is necessary after a hash engine update to maintain the availability of sealed data. Data resealing of remote data is necessary if the PCR values change, but the system is still in a trusted state. This usually happens only when software updates were performed because the software updates do not modify the underlying hardware of the system. As the sealing function relies on the content of PCRs, we propose a procedure to reseal data to the updated PCR values that represent the current state of the system. To understand this procedure we review the sealing protocol in brief.

$$reseal \Leftrightarrow unseal(Hash_{old}) + seal(Hash_{new}) \tag{3.2}$$

Notation 3.2 shows that prior to sealing the data with the new hash engine ($Hash_{new}$), it has to be unsealed first by using the outdated hash function ($Hash_{old}$). However, after the SHA-1 update, the PCRs contain the measurements of the system state with the new hash engine. Therefore, to perform resealing, the current system state has to be made available in both representations computed by $Hash_{old}$ and $Hash_{new}$, respectively. To achieve this, along with the two hash engines, two sets of PCRs, are needed to store both representations (old and new) of the system state. For this purpose, the STPM architecture features the operation of two hash engines in parallel to store both system states in the embedded Flash memory of the microcon-

troller, and thus accelerate the resealing process. After a reboot action, the resealing procedure is performed according to Equation 3.2.

The resealing operation being large to be performed inside the STPM, it is done at the CPU of the system. For this, the CPU must be provided with the corresponding PCRs and the sealing key. The private part of the sealing key is determined by the SRK of the STPM, which along with the sealed data and platform state is sent to the CPU of the system over the LPC bus. Further, the unsealing part of the resealing operation, should be done before loading the STPM with the new hash engine because after loading with new hash engine the old system state is no more available.

Although the data resealing aspect has been considered by Kühn et al. in [59], they concentrate mainly on the hardware and software life cycle as commonly found in enterprises. Their procedures assume that the system state is trustworthy even after a software update, which does not cover the case of a hash engine hardware update that affects the *integrity measurement* process itself. Therefore, to cope with the update of components of the TPM such as the hash engine, the procedures presented in [59] are not applicable instead new dedicated measures as described in this thesis are required.

3.6.1.6 *HMAC*

A TPM command is authenticated using the HMAC function and one of the OSAP/OIAP protocols as specified by the TCG. Considering that the HMAC function uses the hash engine, it has to be fitted to the new hash engine in the case of an update. This adjustment is related to the update of the TCG firmware which may be handled by the related command. In this regard, the *STPM_update* command is an un-authorized command because the compromised SHA-1 means that the HMAC function may no more be used and the compromised RSA implies that the EK is no more valid. However, secure communication to the STPM can be continued using two different approaches. In the first approach, all new messages, which have an HMAC attached, have to use the new hash function to create the authentication code. Running sessions using OIAP or OSAP can be terminated, if the hash function has been exchanged and the session has to be reinitiated afterwards. However, the overhead introduced by this procedure is tolerable because the algorithm update will not happen too frequently.

The second approach is to use the mode of operation introduced by Bellare et al. in [19]. They presented a proof for the usage of HMAC functions even though the hash engine loses its collision resistance property. Therefore, running sessions can still exploit the old hash engine, while newly initiated sessions operate the updated hash engine. The authors in here prove that, the HMAC is a Pseudo Random Function (PRF) under the sole assumption that the compression function is a PRF. This recovers a proof based guarantee since no known attacks compromise the pseudorandomness of the compression function. Further, it also helps to explain the resistance-to-attack that HMAC shows even when implemented with hash functions whose (weak) collision resistance is compromised. They additionally show that an even weaker-than-PRF condition on the compression function, namely that it is a privacy-preserving MAC that is sufficient to establish HMAC as a secure MAC, as long as the hash function meets the very weak requirement of being computationally almost universal.

3.6.1.7 *Signatures*

Signatures are used during the remote attestation process to attest the state of the system to a remote challenger. For this, the PCR values are signed using the AIK and sent to the challenger for verification. In general, signature creation uses the hash engine, as presented in [70], whereas collisions of the hash engine directly translate to forgeries. However, in this case the signatures are created using the mode of operation presented in [49] and therefore are considered to be valid despite the fact that the hash engine is not collision resistant. The authors in here propose to use randomized hashing as the mode of operation for hash functions intended for use with standard digital signatures and without necessitating any changes in the internals of the underlying hash function (e.g., the Secure Hash Algorithm (SHA) family) or in the signature algorithms (e.g., RSA or Digital Signature Algorithm (DSA)). The goal is to free practical digital signature schemes from their current reliance on strong collision resistance by basing the security of these schemes on significantly weaker properties of the underlying hash function, thus providing a safety net in case the (current or future) hash functions in use turn out to be less resilient to collision search than initially thought. Even though the digital signatures are still valid using the above scheme, new messages should be signed using the new hash engine. The signa-

tures have to carry some information showing the algorithm that was used to create them, as done in X.509 [27] certificates. Also, all certificates that are used during the boot sequence of the system might have to be updated to fit the new signature algorithm. Again these operations are related to the update of the TCG firmware which may be handled by the related command.

3.6.2 *Regaining Trust after RSA Engine Update*

3.6.2.1 *TPM Key Hierarchy*

There exists a fixed key hierarchy in every TPM (regardless of its manufacturer) with the fundamental asymmetric keys, i.e., EK and SRK. Utilizing these keys, the TPM internally generates the additional keys required for various operations and stores them outside of the TPM in a secure form. When transferring keys from a TPM to an external memory, a key hierarchy is established (c.f. Figure 2.3). The keys inside the hierarchy are utilized to encrypt and decrypt the user data while performing bind and seal operations. Before the compromise of the RSA, the keys of the key hierarchy are the RSA keys. They all may be compromised after the RSA compromise causing the loss of all the data protected by those keys. Therefore, after replacing RSA with ECC by the update, a completely new key hierarchy is required to be composed.

Thus, building a new key hierarchy implies first generating EK and SRK and later the lower level keys in the hierarchy. There are two different methods to generate a new EK pair. One approach is to generate new key pair at the STPM manufacturer side and then program it into the NVM of the STPM. For this, the engine update like procedure may be deployed between the manufacturer and the system with STPM. Another approach utilizes a specific command provided in the TCG firmware for this purpose. This command, referred to as *TPM_CreateEndorsementKeyPair*, which when executed establishes a new public/private EK pair. For this, it utilizes the asymmetric cryptographic engine and the key generator along with the random number generator of the STPM. Once the EK pair is generated, the SRK pair is generated utilizing the command *TPM_TakeOwnership*. Given the availability of these two keys, the whole key hierarchy is established subsequently. However, in our case we rely on the first approach for loading the fundamental asymmetric key pair on the STPM.

3.6.2.2 *Remote Attestation and Signatures*

In the case of a compromised RSA engine, the signatures generated during the *remote attestation* process become invalid. This is because, a digital signature is a private key based encryption of the hash computed on a message. Therefore, all these signatures have to be recomputed by means of the updated asymmetric engine, which generates a new private key.

3.6.2.3 *Binding and Sealing*

Once a weakness in RSA is discovered, all key material is obsolete along with the data encrypted by it. Therefore, the data, which is bound or sealed to a platform, can not be recovered securely because these operations have used the compromised RSA keys. However, the new data can be protected by taking advantage of the updated asymmetric engine and the new platform state.

3.6.2.4 *Recovery Strategy*

To overcome the aforementioned problems due to compromised RSA, a step-by-step procedure, as mentioned below, has to be followed.

1. Check for an availability of back-up of the data in the system. If not, the data must be restored before updating the engine.
2. Perform an update of the compromised RSA engine with a new asymmetric engine by utilizing the defined update algorithm.
3. The EK is an asymmetric key which must be compatible with the new asymmetric engine. Thus, there is a need to load/generate (depending on the TPM vendor) a new EK. Sign the new EK by a trusted party to indicate that a new asymmetric engine has been loaded onto the STPM.
4. Take ownership of the system to generate a new SRK, which is the root key of the TPM key hierarchy.
5. Generate a new key hierarchy utilizing the new SRK and the new asymmetric engine.
6. Signatures must be generated utilizing the new asymmetric engine for use in remote attestation procedure.

7. Bind/Seal the new data with the new keys of the generated key hierarchy. Further, old data may be secured utilizing the new keys.

In the following, the fulfillment of non-functional requirements (common for both SHA-1 and RSA engines) by the STPM is detailed.

The overall time consumed for performing the update depends on the individual time consumption by the intermediate steps during the update procedure. These steps include transmission of update data from the update server to the PC, through to the STPM, its authentication and decryption by the update algorithm, and at the end configuration of the dynamic region with the new cryptographic engine. Though the first two steps depend on the communication medium (channel and bus), the other steps make use of a lightweight symmetric cryptographic engine (AES) and a fast configuration port (Internal Configuration Access Port (ICAP)).

To cope with the over-provisioning of resources in the dynamic region, it must be not only sized sufficiently for the initial application, but enough unused resources must be left to support future logic configurations. However, this is the critical design planning consideration which is the task of the STPM architecture design team and may be supported by enhanced Computer Aided Design (CAD) tools.

The process of combining the dynamic region with the static region has both technical and economic advantages [110]. Technically, the solution offers a greater performance with lower power dissipation. Further, by keeping static region to dynamic region connections on the same die, valuable static I/O pins are also conserved. Economically, the solution can be less expensive because the dynamic logic fabric does not require any unique semiconductor processing above and beyond the base static logic. In addition the cost of assembly, test, and packaging of a second chip are also eliminated.

Utilizing the proposed generic STPM architecture as the base, a proof-of-concept implementation of the STPM design along with an update of compromised cryptographic engines on it, is detailed in the next chapter.

4

DESIGN AND IMPLEMENTATION OF STPM

With the available description of the requirements and specifications for the STPM architecture design, this chapter details the proof-of-concept implementation of the STPM. In specific, it deals with selecting the components available in the market today and integrating them together to build an STPM architecture such that the resulting design meets all possible requirements mentioned in the previous chapter.

4.1 PLATFORM TECHNOLOGY - FPGA

Considering that the goal of the STPM architecture is to support an update of cryptographic algorithms on it, the underlying platform must be reconfigurable. This is because, reconfigurability provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Given this, FPGAs are one such group of devices that offer the reconfigurability feature by loading different configurations called bitstreams in their terminology. In the following architectural details of *Xilinx* FPGAs is given.

In its architecture, FPGA is a programmable semiconductor device that is based around a matrix of Configurable Logic Blocks (CLBs) connected via programmable interconnects as depicted in Figure 4.1 [85]. As opposed to an ASIC, where the device is custom built for the particular design, FPGAs can be programmed to the desired application or functionality requirements. The CLB, which forms the basic logic unit in an FPGA, consists of a configurable switch matrix with 4 or 6 inputs (Look-Up-Table (LUT)), some selection circuitry (Multiplexer (MUX)), and flip-flops. A high level overview of the CLB with a 4-input LUT, a single flip-flop, a MUX, clock (Clk), and reset (Rst) signal is depicted in Figure 4.2. While the CLB provides the logic capability, flexible interconnect routing routes the signals between CLBs and to/from the I/O. All the three components i.e., logic, routing, and I/O in an FPGA are programmable.

It is the design software that makes the interconnect routing task hidden to the user, thus significantly reducing design complexity. Current FPGAs provide support for dozens of I/O stan-

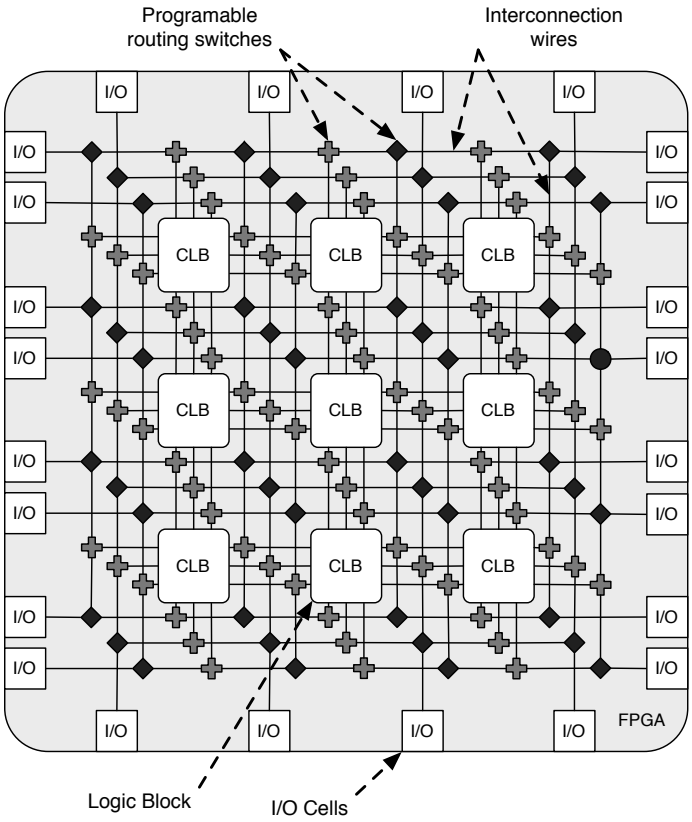

Figure 4.1: Architecture of an FPGA [85]

dards thus providing the ideal interface and flexibility in bridging the system interface. Special memory unit such as an embedded block RAM is available in most FPGAs, which allows for an on-chip storage in the design. Further, dedicated arithmetic blocks such as Digital Signal Processors (DSPs) for fast mathematical computations and high performance microprocessor embedded blocks are supported by some of the high-end FPGAs. A digital clock management which acts as the clocking resource for the design is also supported by the FPGAs. In this context, the Virtex-5 FPGA platform [108] from *Xilinx* supporting powerful features, such as PR (as detailed below), platform Flash Programmable Read Only Memory (PROM), both high and low speed interfaces, is utilized for a proof-of-concept implementation of the STPM design in this thesis.

4.2 PARTIAL RECONFIGURATION (PR)

Partial reconfiguration is the feature supported by some of the high-end FPGAs utilizing which it is possible to provide simul-

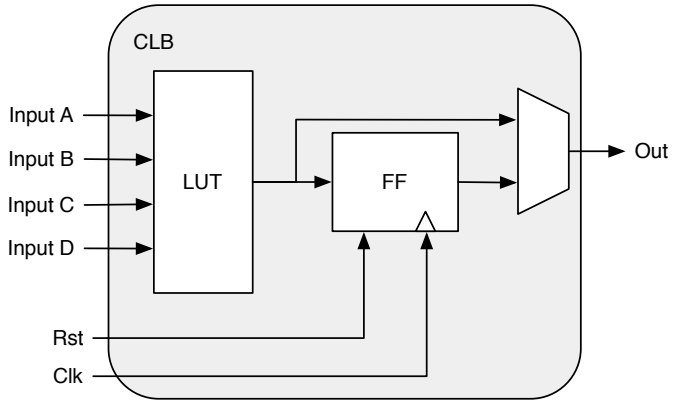

Figure 4.2: Overview of a CLB [85]

taneous availability of both static and dynamic regions in the device logic. Considering that this feature being the fundamental requirement in designing the STPM, details about it are given in the sequel.

There is an analogy between the general-purpose processor's context switching and the PR of an FPGA i.e., the former being the time multiplexing of software resources (programs) and the latter being the time multiplexing of hardware resources (application logic) [105]. Thus, the flexibility of FPGAs, by providing an in-filed update, is taken one step further by the use of PR feature. With PR, not whole FPGA but only parts of it are reconfigured by loading a partial configuration file, called partial bitstream. After creating an operating design by configuring the FPGA with a full bitfile, a partial bitfile may be downloaded to modify pre-defined reconfigurable regions on it. Further, PR ensures the modification of an operating design without compromising the integrity of applications executing on those parts of the device that are not being reconfigured. Thus, with PR, the logic in the FPGA design is divided into two different types, reconfigurable (or dynamic) logic i.e., the portion being reconfigured and static logic i.e., the portion resuming the work. If the configuration of the FPGA is changed at run-time i.e., the system is neither stopped nor switched off, then it is called as dynamic partial reconfiguration.

In the following few definitions are given before explaining the PR process itself. In the PR terminology, the area of the FPGA that is reconfigured is called the Partially Reconfigurable Region (PRR) which typically consists of a number of CLBs and functional blocks. The module to be placed inside the PRR is

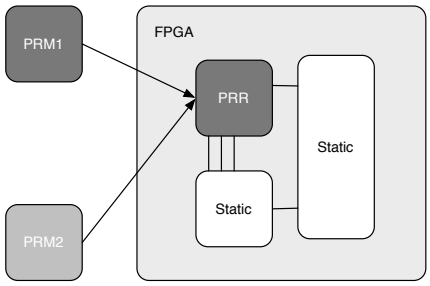

Figure 4.3: Partial Reconfiguration in FPGAs [85]

called a Partially Reconfigurable Module (PRM), which is the specific configuration of the PRR, and at least two PRMs are needed per PRR for a PR design. A macro is a predefined configuration for one or more FPGA components and a bus macro is used for communication between static and dynamic regions on an FPGA.

There are different methods to achieve a PR design such as Difference-based, Module-based, and Relocatable. In a difference-based PR, the new design consists of only the difference between current and new configuration file. In contrast, in a module-based PR design, the partial module contains a new configuration part of an FPGA which is independent of current configuration. In relocatable configuration, the PRM may be moved to a different location on an FPGA without modification. However, in the following we address only the module-based PR because it is the same technique based on which the STPM architecture has been designed.

The PR of an FPGA is illustrated in Figure 4.3. In here, we see that two PRMs which are mutually exclusive in time will be placed in the PRR inside the FPGA i.e., only one PRM can be assigned to a given PRR at a given time. The remaining region in the FPGA which is outside the PRR is the static region, where the application which needs to be run uninterruptedly, is placed. The configuration files placed inside the PRR are called as partial bitstreams and the one placed inside the static region is called the full bitstream. While configuring the FPGA in a PR design, there exists a single full bitstream, in here the hardware logic for controlling the loading of one out of multiple partial bitstreams (PRMs) which are the applications themselves.

An application, Software Defined Radio (SDR), that may utilize the PR feature is detailed in the following. In a SDR system, different wave forms correspond to different radio chan-

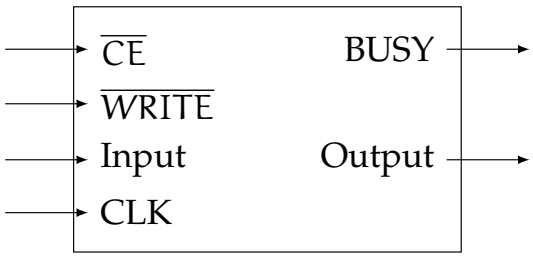

Figure 4.4: Block Diagram of the ICAP [67]

nels with a given wave form establishing a communication with a given channel [54]. These wave forms may be running in the dynamic logic of the FPGA, and the user can upload a new waveform i.e., a partial bitstream, into the dynamic logic to listen to a new channel. Further, any number of waveforms can be supported by a single hardware platform without disrupting the established links to other channels, which is performed by the full bitstream in the static logic of the device. This example application explains the advantage that can be gained by utilizing the PR feature, i.e., reducing the size of the FPGA required for implementing a given function with consequent reductions in cost and/or power consumption.

4.3 CONFIGURATION PORT: ICAP

The ICAP is the one which performs the operations of the configuration port depicted in STPM architecture (c.f. Figure 3.7). The loading of partial bitfiles into the dynamic region of the FPGA is done through the ICAP, a built-in hard core IP module available on the FPGA, which is a part of the static logic. Further, the ICAP interface offers the possibility of the configuration data read out. Unlike the *SelectMAP* [5], the ICAP is an internal part of the FPGA. All *Xilinx* Virtex devices starting from the Virtex-2 have a built-in ICAP and is controlled by the software driver for the processor on the FPGA. In addition to the ICAP, the high-end Virtex-5 provides serial interfaces such as SPI or Joint Test Action Group (JTAG) and parallel interface such as *SelectMAP*, all of which use the same procedure as the ICAP for reading and writing the registers . With the ICAP, it is possible to perform a partial reconfiguration at run time. Here, the bitstream can not be transmitted over the conventional programming interfaces such as JTAG or SPI, therefore the bitstream is first stored in a Block Random Access Memory (BRAM) [107] or external media before configuring the device through ICAP. The other alterna-

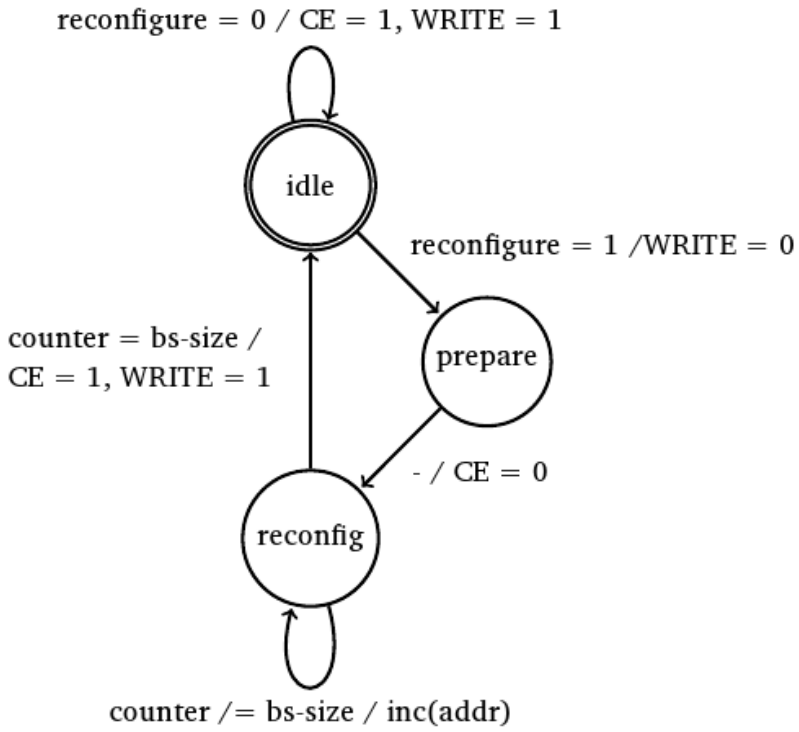

Figure 4.5: ICAP Write Process [67]

tives include the configured access via communication media such as Universal Asynchronous Receiver/Transmitter (UART) and Ethernet.

The block diagram of an ICAP module is depicted in Figure 4.4. It has separate bus for both input and output (denoted by "Input", and "Output" ports), which may be operated with 8, 16 or 32 bit wide data and a clock (denoted by "CLK") frequency of up to 100MHz [9]. The data width is specified at instantiation, and the FPGA adjusts itself automatically, after detecting the bus width sequence. Through the active low "Chip Enable" (*CE*) signal, the interface is activated and a switching between writing and reading is done. The *WRITE* input indicates the direction, whether the data is written (*WRITE* = 0) or read (*WRITE* = 1) from the device. The *BUSY* signal indicates the status of the read requests, i.e., only when it has reached a low level, the data is stable at the output.

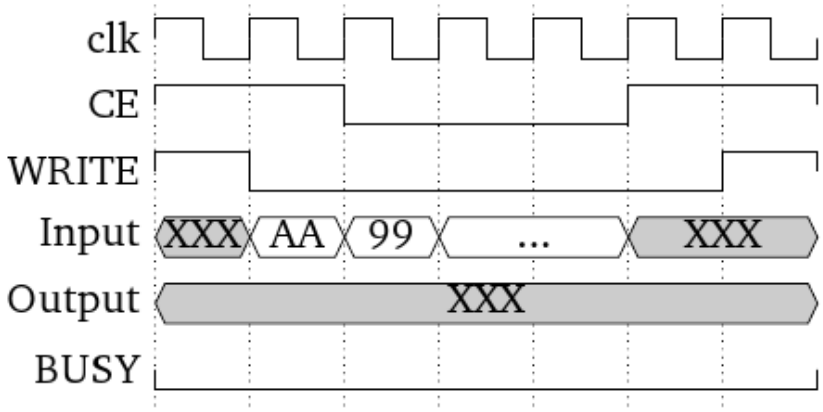

Figure 4.6: Signal Diagram for Write Operation [67]

4.3.1 *ICAP Write Operation*

A state machine illustrating the write process of the ICAP is depicted in Figure 4.5. In the 32-bit bus configuration, per every rising clock edge, the ICAP expects a data word at its input, where as in the 8-bit mode, the data word corresponding to a byte is split into four clocks. The protocol calls for a change between read and write modes that the Chip Enable (CE) signal is deactivated before the write signal. After the change of direction by the CE, the ICAP must be reactivated.

The state machine remains in the ”idle" state until a reconfiguration command is given. Once the reconfiguration activation signal becomes stable at the input, the state machine prepares to switch to ”prepare" state and sets the direction of writing the data. In the following state i.e., ”reconfig", the ICAP is activated and processes the input with each rising edge of the clock. Therefore, the input bitstream has to be previously stored in a buffer or it must be ensured that the clock speed is adjusted according to the data stream. If the source data can not be delivered fast enough, it is possible to interrupt the writing process and to continue it later. For this, the CE input is set to high to stop the ICAP operation or the clock signal is interrupted, which also brings the ICAP to a halt. Once the CE goes low again and the clock runs, the ICAP takes new commands. The waveform depicted in Figure 4.6 shows the sequence of a write operation and the points in time when the CE or *WRITE* signal must be set.

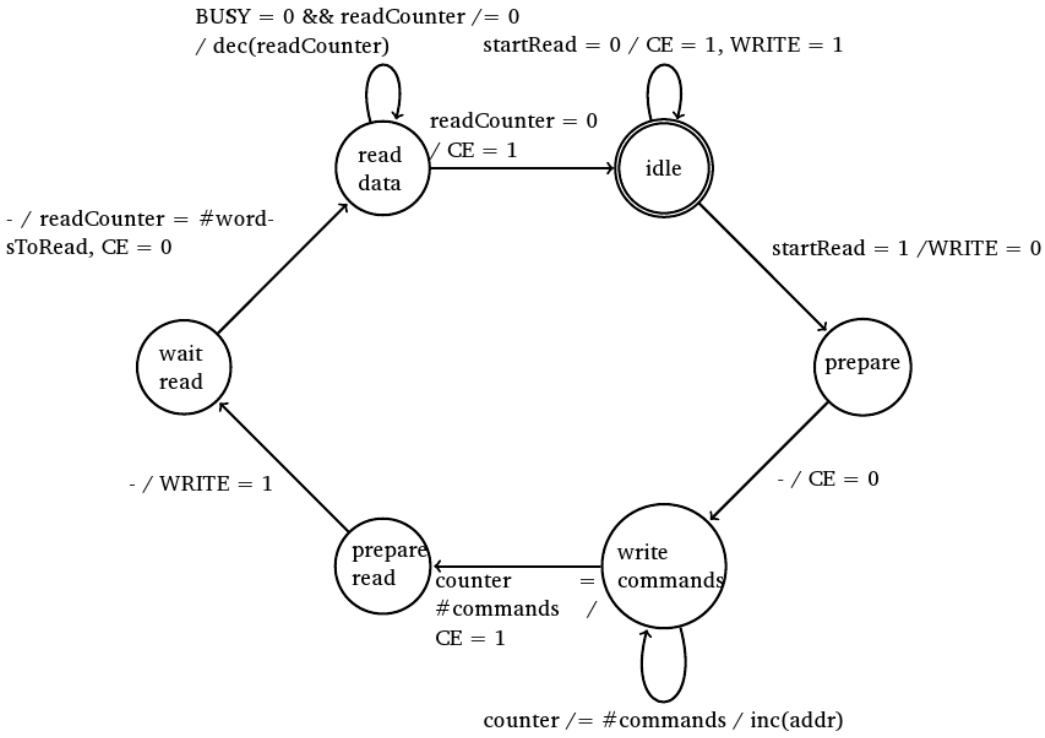

Figure 4.7: ICAP Read Process [67]

4.3.2 *ICAP Read Operation*

To read a register via the ICAP, as done in write process, the read command is sent and consequently the *WRITE* signal is changed to the read mode. This is represented as the state transition ”idle" -> ”prepare" -> ”write commands" -> ”prepare read" in Figure 4.7. It must be ensured that the ICAP is deactivated first (i.e., CE = 1) and then the direction is changed (*WRITE* = 1). With renewed activation of the ICAP the data is started to be sent via the output. This data can be read when the *BUSY* signal is set to low (see Figure 4.8). Depending on which register is defined for readout, one or several words are stored in it. Illustrated in Figure 4.7 is only a simplified transition from the ”read data" state to the ”idle" state but in a real application, commands arrive at the ICAP that need desynchronization and a Cyclic Redundancy Check (CRC) reset.

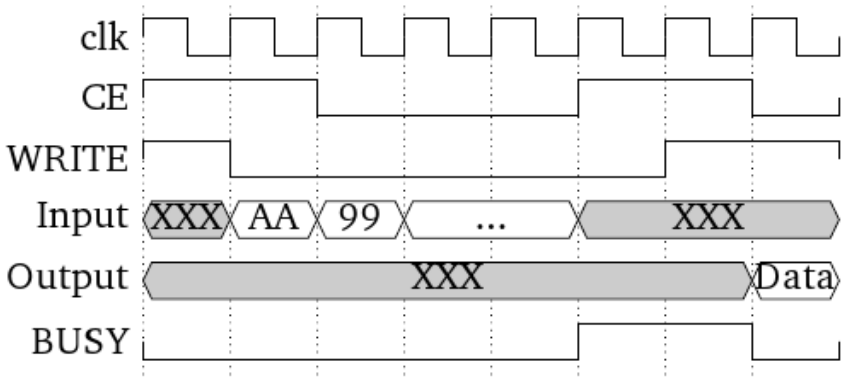

Figure 4.8: Signal Diagram for Read Operation [67]

4.4 EXECUTION ENGINE: MICROBLAZE BASED MICROCONTROLLER SYSTEM

As mentioned before, the execution engine of the STPM is a microprocessor with additionally attached components such as memories and I/O for executing the TPM commands. For this, a *MicroBlaze* processor based microcontroller system as depicted in Figure 4.9 is utilized in here. The depicted system is highly integrated and standalone which is intended for controller applications. Data and program are stored in a local memory while the debug is facilitated by the Microblaze Debug Module (MDM) (an optional feature shown in gray). A standard set of peripherals is also included, providing basic functionality such as the interrupt controller, UART, timers and general purpose I/O.

The *MicroBlaze* embedded processor is a soft core processor with a Reduced Instruction Set Computer (RISC) optimization for implementation in *Xilinx* FPGAs [103]. Local memory is used for data and program storage and is implemented using BRAM. The size of the local memory is parameterized and can be between 4kB and 64kB. The local memory is connected to the *MicroBlaze* through the Local Memory Bus (LMB) and the LMB BRAM interface controllers. The MDM, connects *MicroBlaze* debug logic to the Xilinx Microprocessor Debugger (XMD), which is a low level debugger. The XMD can be used for downloading software, to set break points, view register and memory contents. The I/O Module is a light-weight implementation of a set of standard I/O functions commonly used in a *MicroBlaze* processor sub-system.

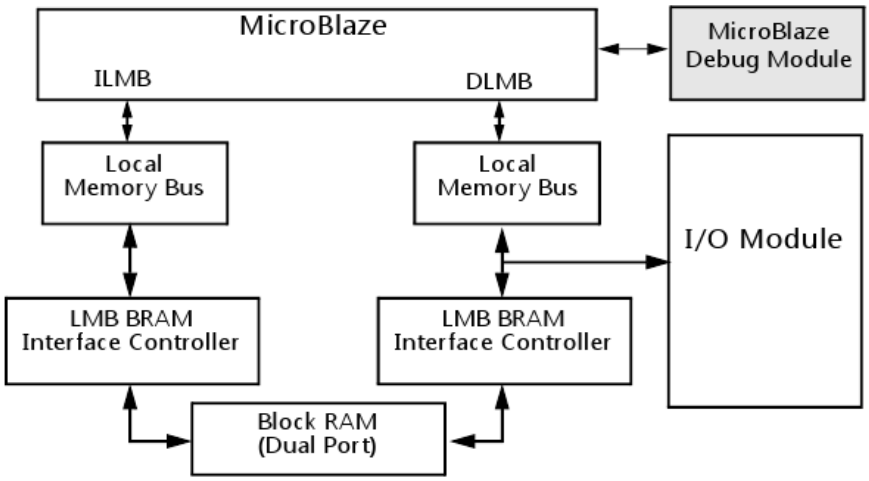

Figure 4.9: MicroBlaze based Microcontroller System [104]

The *PetaLinux* [97] is a *Linux* distribution and development environment designed specifically for embedded systems, realized using the *Xilinx MicroBlaze* soft-core processor. *PetaLinux* contains the custom code of the Linux *kernel*, µClibc and a corresponding adaptation of *GNU* tool chain (assemblers, compilers for *C*, *C++*, linker, etc.) for developing the Linux *kernel*, libraries, and programs. On this loaded distribution, the processor of the microcontroller executes the TPM command execution software, referred to as the *TPM emulator* [86]. It is a software written in *C* language that emulates a conventional TPM with its command set and functions according to the TCG specifications. It is implemented as *daemon*, that reads the native TPM commands in the form of sequences of bytes over the *Unix* domain socket interface and sends back the TPM responses. The data that must be stored permanently are stored in a file, similar to the way the hardware TPMs store in a secure storage on the chip. The cryptographic operations included with the *TPM emulator* are the RSA, SHA-1, HMAC, and for the random number generator the system command is used. For the system, the *TPM emulator* appears just like a hardware TPM, which would be addressed in the same way.

Thus, utilizing the aforementioned internal components, i.e., the *MicroBlaze* processor and *PetaLinux* distribution with the running *TPM emulator* on it, along with memory and interfaces, the microcontroller sub-system provides all the functionalities as required by the execution engine of the STPM.

4.5 NVM INTEGRATION ON THE STPM

As mentioned in the previous chapter, one fundamental requirement of the STPM architecture is to hold an on-board NVM for storing persistent data such as cryptographic keys for securely storing the platform data. However, the chosen platform in our case is an SRAM based FPGA, which does not support such a persistent memory and furthermore, the device itself is volatile in nature. Thus, there is a need to integrate an external NVM programmed with keys and then access it accordingly. For such an integration and access, it is relied on the mechanism proposed by Schellekens et al. [81], as detailed in the following.

4.5.1 *Outline of NVM Access Protocol*

The authors in here integrated a multiple-time-programmable NVM onto an embedded System-on-Chip (SoC) for building a trusted computing system. The NVM is protected against non-invasive attacks utilizing a minimal cryptographic protocol that establishes an authenticated channel between the system and the NVM. The NVM access protocol relies on "challenge-response" pairs generated utilizing a PUF [87, 20]. The PUF is a physical structure that is present on an integrated circuit due to variations in the manufacturing process. It uniquely identifies a given device and the mathematical function derived from it can not be cloned on another device. Additionally, a shared secret K_{Auth} between the NVM and the FPGA, also derived from the PUF is required by the access protocol. Then utilizing a MAC based authentication scheme a secure communication channel is established between the external NVM and the FPGA of the STPM.

A PUF may be implemented with the CLBs of the FPGA, referred to as "Delay PUF", or with the start-up values of an uninitialized BRAM, referred to as "SRAM PUF". The key, K_{Auth}, is extracted from the corresponding PUF response, R_{Auth}, at the point in time when needed, using the fuzzy extractor and corresponding helper data. The corresponding challenge, C_{Auth}, is embedded in the FPGA's bitstream and moreover, the PUF guarantees that given the challenge its response is still unpredictable.

There exists a commercially available product, Quiddikey™ from Intrinsic-ID™, providing a secure key storage functionality utilizing the PUF [52]. It extracts the key derived from the

PUF to protect the device and its content from counterfeiting and cloning. It is available as an IP core and can be embedded into any chip design. The advantage here is that the key is not present when the device is powered-off. Given this, the K_{Auth} required in the NVM access protocol may be derived from this product.

4.5.2 *Protocol Operation*

Read and write operations on the NVM, utilizing this access protocol are illustrated in the following.

To describe the protocol operation we define the following notation:

r	Random number acting as a nonce for the FPGA
i	The address location inside the NVM
c	Monotonic counter value acting as a nonce for the NVM
M_i	Message at address i
K_{Auth}	Shared secret key for generating the HMAC
$H_{K_{Auth}}(c+1)$	HMAC of counter $c+1$ computed with key K_{Auth}

The FPGA module uses random numbers as source of freshness in the protocol, while the NVM makes use of a monotonic counter c. Alternatively, the NVM may use a random number generator to generate fresh nonces. The data read operation by the FPGA of the STPM from an address location i of the NVM is depicted in Figure 4.10. For this, first the FPGA generates a nonce r and sends it to the NVM, together with address i. Then the memory reads the content at address i, i.e., message M_i, and the counter c from the internal address 0. Later, the memory returns M_i accompanied with c and a MAC on r, i, M_i, and c. The FPGA accepts the message M_i and performs consequent operations, only if the verified MAC is OK, otherwise it stops all operations. Further, it stores c in volatile memory, because this acts as a challenge for a subsequent write operation.

In order to write data securely to the NVM, the write operation by the FPGA consists of the following steps (c.f. Figure 4.11). The FPGA retrieves c from its internal memory which

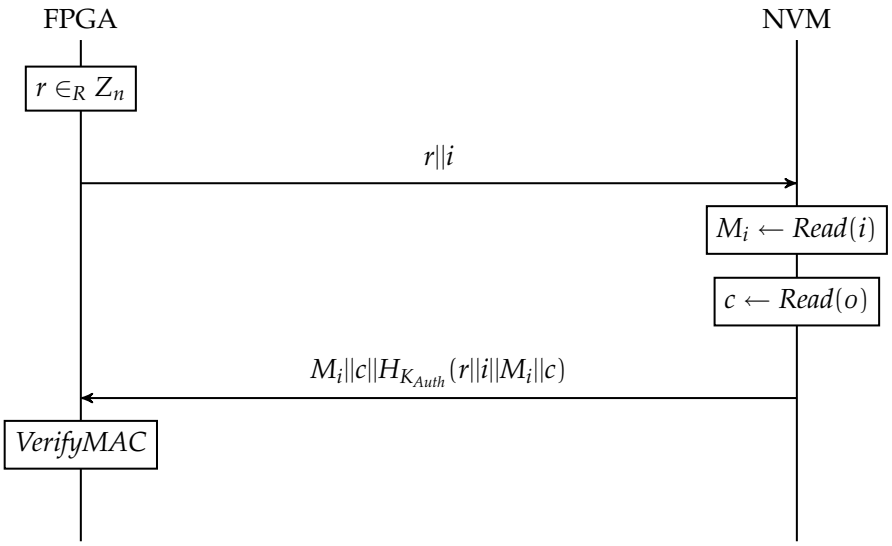

Figure 4.10: Read Protocol [81]

was stored during a previous operation. It sends the message M_i'', that it wants to write at address i, accompanied with a MAC on i, M_i'', and c. The NVM reads the monotonic counter c from the internal address 0 before checking the MAC. If the MAC is OK, it implies that the FPGA used the same value of c. Then it stores the data M_i'' at i and increments the monotonic counter by writing $c + 1$ to internal address 0. The memory returns a MAC on the new counter value (i.e., $c + 1$), which is verified by the FPGA to determine whether the write operation was successful or not.

4.6 PROGRAMMABLE FLASH

In addition to the NVM described above, the STPM architecture must also contain an on-board rewritable NVM such as programmable Flash to store the bitfiles that need to be loaded into the static and dynamic regions on every boot-up of the system. This Flash is programmable because it has to support loading of new bitfiles (i.e., new cryptographic engines) as required by the user. In the generic architecture of the STPM, such a memory was included inside the execution engine as embedded Flash (c.f. Figure 3.9). However, the utilized FPGAs being volatile, the bitfiles are stored in an external Flash memory. A high performance interface, referred to as *SelectMAP* configuration interface, is utilized to access the data from this Flash and

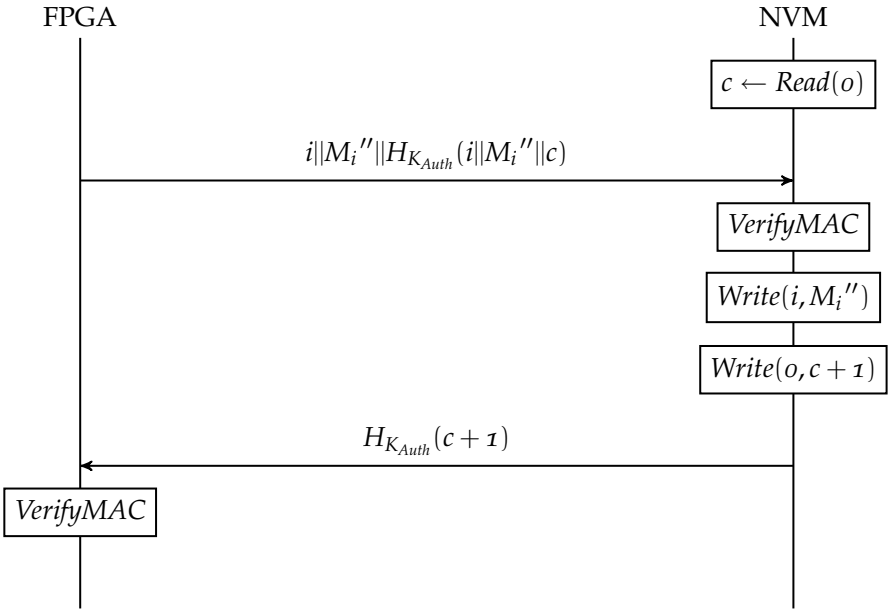

Figure 4.11: Write Protocol [81]

to configure the FPGA. The *SelectMAP* configuration interface provides an 8-bit, 16-bit, or 32-bit bidirectional data bus interface to the Virtex-5 configuration logic that can be used for both configuration and readback [107].

In this context, the Platform Flash XL is the memory that is specially optimized for high-performance *Xilinx* Virtex-5 FPGA configuration and ease-of-use. This memory integrates 128 Mb of in-system programmable Flash storage and performance features for configuration within a small-footprint FT64 package. Power-on burst read mode and dedicated I/O power supply enable Platform Flash XL to meet seamlessly with the Virtex-5 FPGA *SelectMAP* configuration interface. A wide, 16-bit data bus delivers the FPGA configuration bitstream at speeds upto 800 Mb/s without wait states. A simplified model of the Platform Flash XL configuration solution for a Virtex-5 FPGA is depicted in Figure 4.12.

Although two different types of memories i.e., an external authenticated NVM and a programmable Flash for storage purpose are presented here, they may reside in a single physical component while manufacturing the STPM as a product.

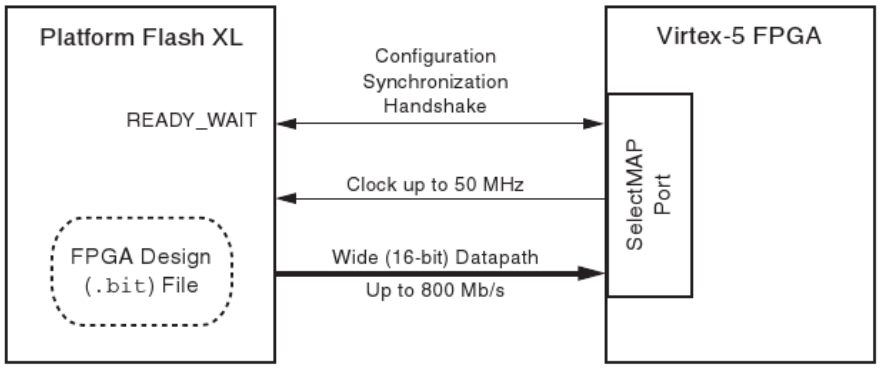

Figure 4.12: Flash XL High-Performance FPGA Configuration [107]

4.7 DESIGN OF THE STPM ARCHITECTURE

Based on the aforementioned selected components, a novel architecture for the STPM utilizing an SRAM based FPGA is depicted in Figure 4.13. The TPM communicates with the central processor of a PC over an I/O interface i.e., the LPC bus which is standardized by the TCG. The other bus interfaces could be System Management Bus (SMBus) and Advanced Microcontroller Bus Architecture (AMBA) bus, which essentially provide a secure exchange of data between the processor and the TPM. Thus, to comply with the standards, the STPM is also required to communicate over one of the above interfaces, in this case the LPC bus. However, when the STPM is used in an embedded system the user is free to choose any available SoC bus.

As mentioned before, the underlying FPGA architecture of the STPM supports the PR feature. Utilizing this feature the FPGA's logic is divided into static and dynamic parts as depicted in Figure 4.13. The dotted lines separating the regions denote the communication boundary between the two parts. The static logic of the FPGA is loaded with the non-updatable components, i.e., a full bitfile from the Flash, upon initialization of the device. This full bitfile comprises of *MicroBlaze* based microcontroller system (execution engine) and the update algorithm of the STPM. Though the configuration port (i.e., ICAP) is included as a part of the update algorithm (c.f. Figure 4.13), it is available as a hard-core IP on the FPGA. However, its inclusion in the update algorithm is to illustrate the logic resource consumption of such a port, as done later in this chapter while describing the implementation of the STPM architecture. The execution engine executes all the TPM commands given by the processor

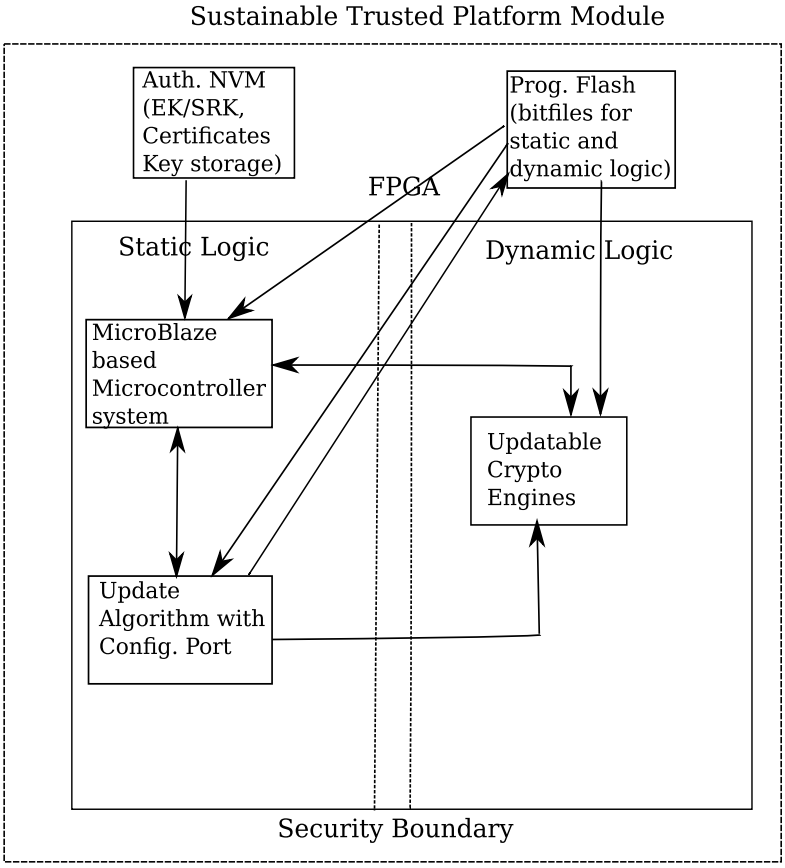

Figure 4.13: Design of STPM Architecture based on an FPGA

and the results of these commands are communicated over an I/O interface. The update algorithm provides a secure update of the cryptographic engines on the STPM, arriving as bitfiles from the server. In addition to the ICAP interface, the update algorithm consists of a PR controller for controlled loading of the engines into the dynamic part. The dynamic part of the FPGA is loaded with partial bitfiles, which represent the cryptographic engines of the TPM. Additionally, this part being reconfigurable, allows the update of the loaded engines when they are compromised. The communication between the NVM, Flash, and the FPGA (along with its internal communication) utilize the aforementioned protocols and interfaces accordingly. Further, the STPM (with all its components) is protected by a secure package, referred to as security boundary, to avoid physical attacks on it. The most important feature of the STPM is to securely support hardware update (i.e., cryptographic engine update), thus providing a flexible and scalable design of cryptographic en-

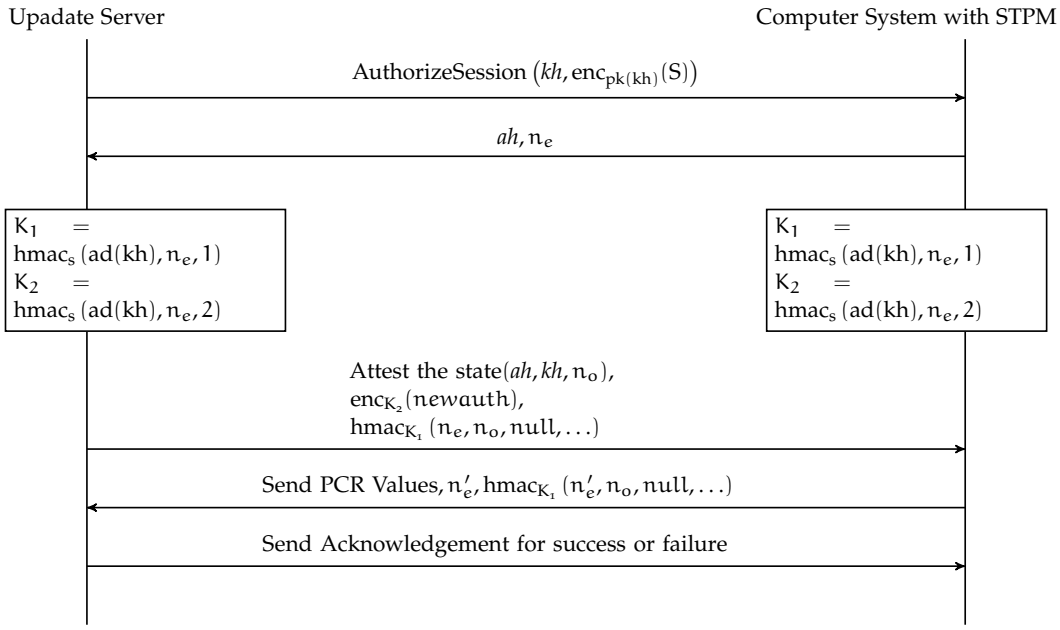

Figure 4.14: Protocol for Secure Session Establishment [40]

gines on it, which is in general not the case with conventional TPMs.

4.8 SESSION ESTABLISHMENT PROTOCOL

A secure session needs to be established between the update server and the PC (with embedded STPM inside it) before performing an update. For this, a well-defined communication protocol, called SKAP (c.f. Figure 4.14), designed by Chen et al. [26] has been utilized. The designers of SKAP utilized the protocol for a secure session establishment between the user process and a conventional TPM, for executing commands and for message exchange. Therefore, this protocol has been adopted to the STPM requirements as defined by Feller et al. in [40]. In specific, unlike the standard approach the modified protocol is utilized for communication between an entity supporting partial reconfiguration and a remote entity. This scenario is identical to the considered scenario here i.e., communication between the STPM and the update server. Thus, we rely on the same protocol for a secure communication between these two entities. However, it is the PC, which communicates directly with the update server, through an external channel, to obtain the update data before transferring it to the STPM. Furthermore, the communication be-

tween PC and the STPM is assumed to be trusted to avoid any additional problems created by a compromise of this internal channel.

As a first step in the session establishment process, a command for authorizing the session, referred to as *AuthorizeSession*, with a key handle kh and an encrypted secret S (utilizing kh) as parameters, is sent from update server to the PC. In response, the PC sends an authorization handle ah and a newly generated nonce n_e back to the update server. With the available secret S, nonce n_e, and handle kh both the entities compute two secrets K1 and K2 on their side. These secrets are the HMAC computations on the handle kh and nonce n_e utilizing the secret S as the key. Then the command, *AttestState* with a set of parameters, requesting the attestation of the computer system is issued by the update server. In turn, the PC responds to the server with a digitally signed set of PCR values to authenticate both, itself and the STPM, to the server. If the authentication is successful then a session is established between the server and the STPM, else the session is terminated. The success or failure of the update process is reported to the server by means of additional authentication data. After establishing a secure session, the update phenomenon follows the steps defined in Section 3.5.2. Unlike OSAP and OIAP protocols specified by the TCG in [92], SKAP relies on public key cryptography to avoid both, TPM impersonation and weak authorization data attacks as presented in [26].

4.9 UPDATE UTILIZING THE STPM ARCHITECTURE

With the available STPM architecture and the protocol for secure session establishment, an update of two of the fundamental cryptographic engines, i.e., the SHA-1 and the RSA, utilizing the defined update algorithm are detailed as test cases in the following [64, 65].

4.9.1 *Test Case1: Update of the SHA-1 Engine*

An update of compromised SHA-1 engine with an uncompromised new SHA-2 engine is depicted in Figure 4.15. The set up is identical to the system level view of an algorithm update on the STPM as depicted in Figure 3.2. The update server comprises of the bitfile library with the configuration data (denoted by PRMs) for the defined locations (denoted by PRRs) inside the dynamic

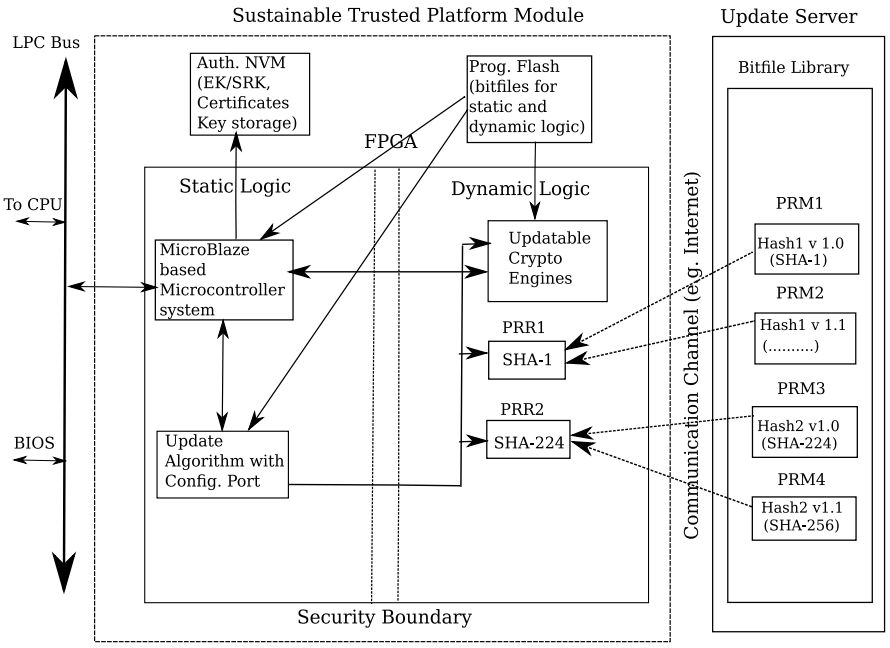

Figure 4.15: Updating SHA-1 Engine on STPM

logic of the FPGA. The dotted lines extending from the PRMs in the update server to the PRRs inside the dynamic logic indicate an association of cryptographic engines to their corresponding locations. Though it is depicted as a direct communication between the update server and the STPM for simplicity, the actual procedure for configuring the dynamic logic is performed by the host PC through the update algorithm as detailed in Section 3.5.2.

With reference to re-establishing the trust in the system after the SHA-1 engine update (c.f. Section 3.6.1), it is mandatory to unseal the data before resealing it to the new platform. To achieve this, it has been assumed that the dynamic logic has to support execution of two hash engines in parallel i.e., an old hash engine for unsealing the data and a new hash engine for sealing the data to the new platform state. In this context, it can be seen that the dynamic logic of the FPGA in Figure 4.15 holds two PRRs, one with the old SHA-1 engine and the other with the new SHA-2 engine.

On a normal start-up of the PC, the STPM embedded inside it is configured with the initial configuration and is running. This initial configuration, stored inside the Flash memory, is the previous bitstream data for both static and dynamic logics of the

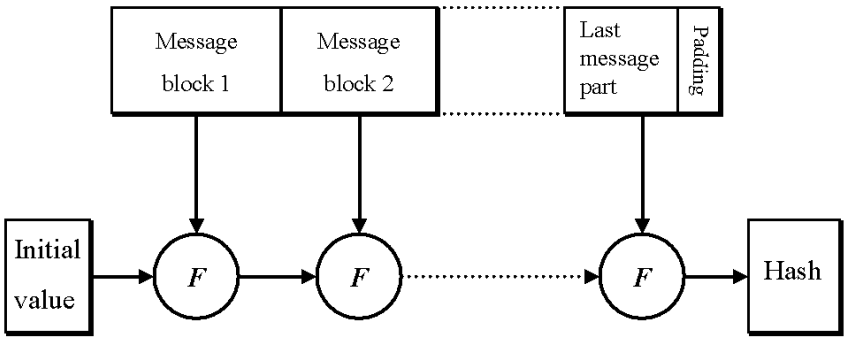

Figure 4.16: Block Diagram for Hash Computation [33]

FPGA. Assuming that the SHA-1 engine on the STPM is compromised, then an update request is sent by the update server to the PC. The communication between the two may be over the Internet in case of a remote update or through an UART controller, in case of a local update. Then, a secure session is established between the PC and the update server utilizing the SKAP protocol (c.f. Section 4.8). Once a secure session is established, the update of the new engine is performed following the procedure defined in Section 3.5.2.

4.9.2 *SHA-1 and SHA-2 Operation*

The most widely used hash functions today belong to the SHA family [33], which are developed and published by NIST. The generalized operation of a hash function (either SHA-1 or SHA-2) is depicted in Figure 4.16. The function "F" represents the core of the algorithms, i.e., the corresponding compression function utilized by one of them.

In the following the individual steps in the operation of the SHA-1 function are given:

- As a first step, the message M of l bits is padded with a value k to give the smallest solution of the expression, $l + k \equiv 448 \bmod 512$, before appending a 1 followed by $k - 1$ number of "zeros" at the message end due to the function being big-endian in operation [43].
- Following this, a 64-bit binary representation of the original length of the message (i.e., l bits) is appended. This padding makes the message M, a multiple of 512 bits, allowing it to be split into N 512 bit blocks.

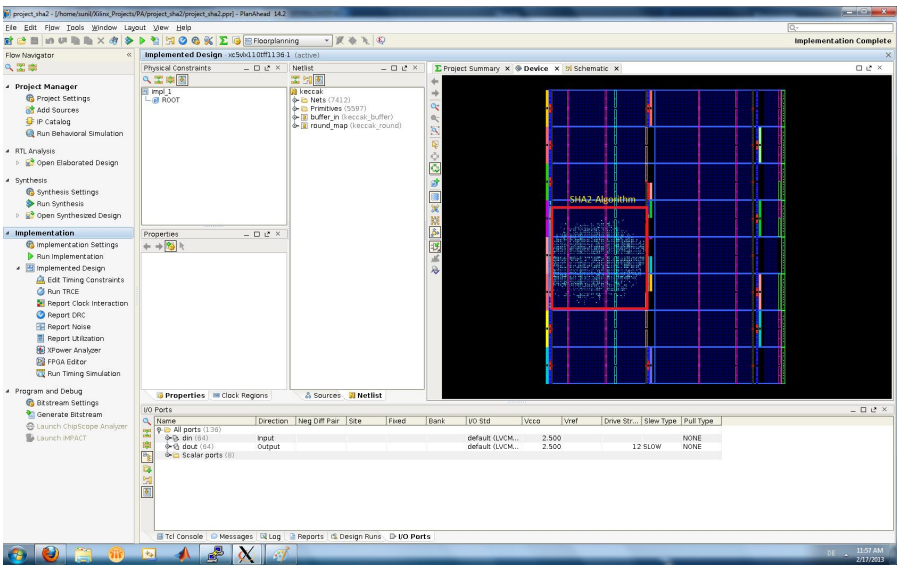

Figure 4.17: FPGA Editor View of the SHA-2 Implementation

- These blocks are then passed through the SHA-1 compression function, which comprises of logical operations such as *XOR* and *LeftRotations* (RotL), to carry out 4 rounds of 20 steps each (giving 80 steps in total) to create a digest.
- While doing this, the hash value will change due to the previous values calculated in a round, thus a buffer is used to hold the intermediate and final result of the hash function, which is initialized with a pre-defined constant, initialization vector (IV). This buffer is 160 bit long which is the length of the digest for SHA-1 algorithm.
- Once all the 512 bit blocks have been processed a 160 bit output is produced, which is the hash value of the input message M.

The SHA-2 algorithm was developed due to weaknesses, i.e., a possible theoretical collision attack, found in SHA-1 algorithm [98]. An implementation of SHA-2 algorithm with 224 bit digest output on a *Xilinx* Virtex-5 LX110T FPGA platform is depicted in Figure 4.17. In its operation, SHA-2 is identical to SHA-1 but because the output digest being one of 224/256/384/512 bits, the primary differences are more number of initial blocks of input message and fewer number of rounds in the compression function [68]. The other differences are the use of right shifts as well as left shifts, constants for each round instead of blocks of

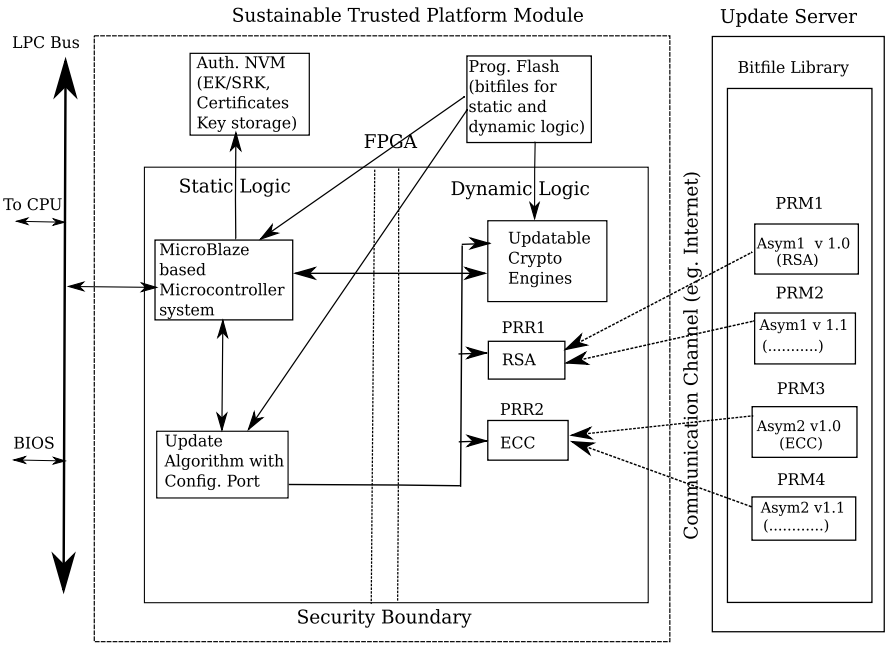

Figure 4.18: Updating RSA Engine on STPM

rounds, and the use of 64 bit inputs on the 512 bit function over the 32-bit use in the former.

- Messages are initially padded from 16 to 64 rounds using logical calculations based on the input message and each round is calculated using six variables based on the eight message blocks and 64 rounds.
- Unlike SHA-1, where there are four round calculations, SHA-2 uses only one round. Its inherent strength comes from the use of 64 round constants over the 4 round constants in the former, which further reduces the risk of collisions.

4.9.3 *Test Case2: Update of the RSA Engine*

An update of compromised RSA engine with an uncompromised new ECC engine [57] is depicted in Figure 4.18. The procedure is identical to the one described in SHA-1 engine update except for the updated algorithm. Once a successful update is performed then the trust in the system is re-established utilizing the steps detailed in Section 3.6.2.

4.9.4 *RSA and ECC Operation*

The algorithms RSA and ECC are public key algorithms, in which each user or the device taking part in the communication have a pair of keys, a public key and a private key, and a set of operations associated with these keys to provide cryptographic capabilities. Only the particular user knows the private key whereas the public key is distributed to all users taking part in the communication. Public key algorithms, unlike private key algorithms (e.g., AES), does not require any shared secret between the communicating parties but it is much slower than the latter.

The RSA algorithm is one of the best known and widely used public key algorithm for protecting the user data. Its operation mainly involves key generation, encryption, and decryption operations as detailed in the following [95].

- For key generation, it first selects two distinct random prime numbers, for example, p and q to compute the product $n = pq$, where n is referred to as modulus.
- Then it computes the Euler's totient function, $\phi(n) = (p-1)(q-1)$ before choosing an integer e, satisfying $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$, where gcd represents the greatest common divisor operation. The chosen e represents the public encryption key and is denoted as (e, n).
- For computing the private key d, denoted by (d, n), an operation, $d \equiv e^{-1} \bmod(\phi(n))$, is performed. Once the public/private key pair is available the encryption and decryption operations on the message may be performed.
- To encrypt a message m, the sender first has to obtain the recipient's public key, (e, n) before representing the message as an integer in the interval $[0, n-1]$. Then compute the cipher text, c, using the operation, $c = m^e \bmod(n)$. To decrypt the cipher text c, the recipient uses his private key (d, n) to compute $m = c^d \bmod(n)$.

The security of the RSA is based on the fact that it is far more difficult to factorize a product of two primes than it is to multiply the two primes. The signature generation and verification operations inherently involve encryption and decryption operations but with private and public keys respectively. An implementation of the RSA modular exponentiation operation based

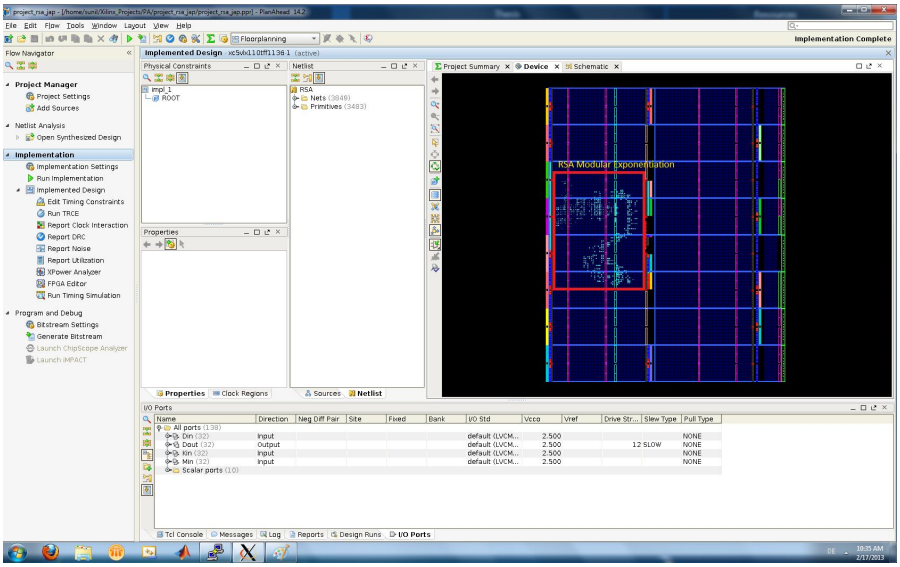

Figure 4.19: FPGA Editor View of the RSA (Modular Exponentiation) Implementation

on 2048 bit key pair, on a Virtex-5 LX110T FPGA platform, is depicted in Figure 4.19.

The operational details of the ECC algorithm are detailed in the following [16]. The mathematical operation of ECC is defined over the elliptic curve $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \neq 0$, where each value of a and b gives a different elliptic curve. All points (x, y) which satisfy the above equation and a point at infinity lie on this elliptic curve with the public key being a point on the curve and the private key being a random number.

The security of ECC depends on the difficulty of elliptic curve discrete logarithm problem as detailed below. Let P and Q be two points on an elliptic curve such that $kP = Q$, where k is a scalar. Given P and Q, it is computationally infeasible to obtain k, which is the discrete logarithm of Q to the base P. Thus, the fundamental operation involved in ECC is point multiplication, i.e., multiplication of a scalar k with any point P on the curve to obtain another point Q on the curve. In contrast, the fundamental operation utilized by the RSA engine is the modular exponentiation, which is computationally less efficient compared to the point multiplication. This feature makes the ECC engine to consume fewer resources on the considered architecture and to operate at a faster speed than RSA. The main advantage of ECC is that it provides the same level of security as provided by

the RSA, but with considerably smaller key sizes. For example, the 1024/2048 bit security strength of the RSA may be offered by only 160/224 bit ECC [72].

Point multiplication is achieved by two basic elliptic curve operations, point addition and point doubling. Point addition is to add two points J and K on the curve to obtain another point L, i.e., $L = J + K$. Whereas point doubling is to add a point J to itself to obtain another point L, i.e., $L = 2J$. For example, let P be a point on an elliptic curve and k be a scalar that is multiplied with the point P to obtain another point Q on the curve, i.e., to find $Q = kP$. Thus, if $k = 23$, then the point multiplication operation $kP = 23.P = 2(2(2(2P) + P) + P) + P$, represents the utilization of point addition and point doubling repeatedly to find the result. The elliptic curve operations on real numbers (as defined above) are slow and inaccurate due to round-off errors. However, the cryptographic operations need to be faster and accurate, thus to make operations on elliptic curve accurate and more efficient, the curve cryptography is defined over two finite fields, i.e., prime field, *GF(p)* and binary field, $GF(2^n)$ [31]. The field is chosen with finitely large number of points suited for cryptographic operations and in here we choose the prime field. The equation of the elliptic curve on a prime field *GF(p)* is $y^2 \bmod p = x^3 + ax + b \bmod p$, where $4a^3 + 27b^2 \bmod p \neq 0$. Here the elements of the finite field are integers between 0 and $p - 1$ and all the operations involve integers between this range. The prime number p is chosen in the range between 112 to 521 bits such that there is finitely large number of points on the elliptic curve to make the cryptosystem secure.

Signature algorithm is utilized for authenticating a device or a message sent by that device. For example, consider two devices A and B, to authenticate a message sent by A, the device A signs the message using its private key. Then it sends the message and the signature to the device B, which can be verified only by using the public key of device A. Given that the device B knows A's public key, it can verify whether the message is indeed sent by A or not. Similarly, Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the DSA that operates on elliptic curve groups. For sending a signed message from A to B, both have to agree upon a set of elliptic curve domain parameters such as n and G. Sender A has a key pair consisting of a private key dA (a randomly selected integer less than n, where n is the order of the curve) and a public key $QA = dA * G$, where G is the generator point.

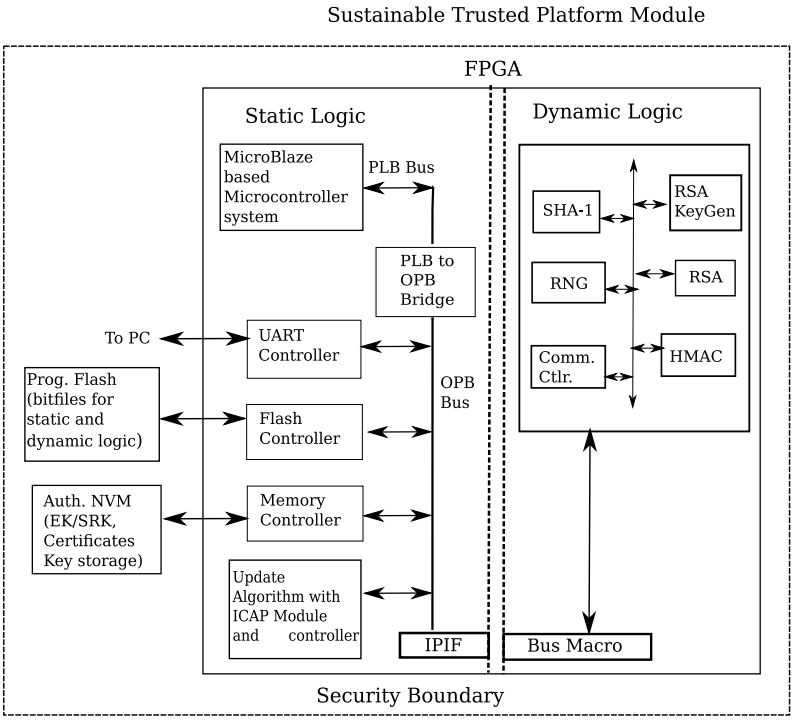

Figure 4.20: FPGA based STPM Architecture Implementation [77]

However, the TCG specification mandates the use of RSA as the standard asymmetric engine inside the TPM. Given the capabilities of the ECC such as performing identical operations as RSA but with more efficiency, in case of a compromise of the latter on STPM, the former is a promising candidate to replace it.

4.10 IMPLEMENTATION OF THE STPM ARCHITECTURE

The STPM architecture is implemented as a proof-of-concept on an SRAM based FPGA as depicted in Figure 4.20. The FPGA utilized is the *Xilinx* Virtex-5 LX110T device supporting PR and the required interfaces as detailed earlier. For the implementation in here, a local update scenario is considered i.e., the PC acts as an update server and the FPGA acts as the STPM module and the communication between them is achieved over an UART interface. In specific, individual components in each of the static and the dynamic regions are formally described as a structure in a hardware description language i.e., VHSIC Hardware Description Language (VHDL), for performing an analysis.

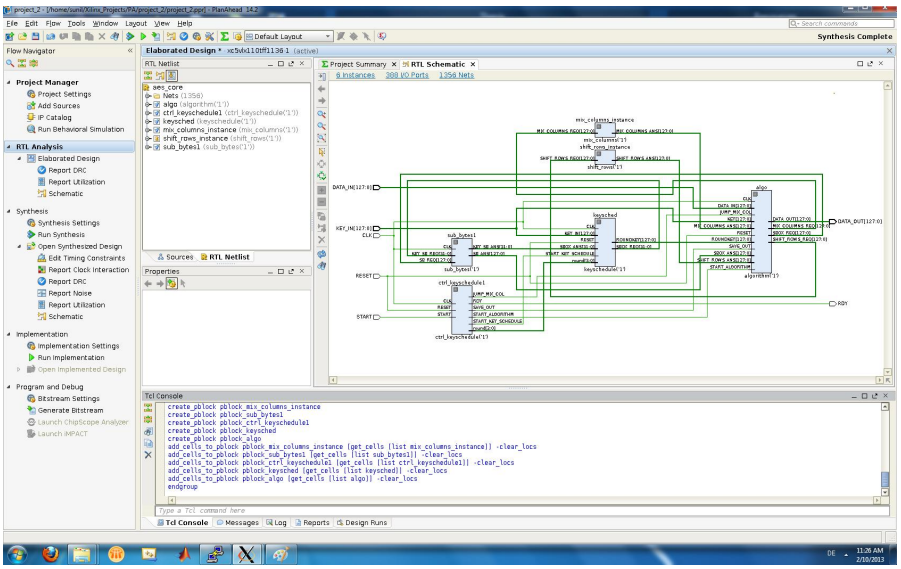

Figure 4.21: RTL Schematic of the AES Algorithm

This structural description of each of the block is first simulated and then synthesized utilizing the vendor specific tools for measuring their performance in terms of execution speed and computational resources consumed on the target device as detailed in the following.

In specific, the components execution engine is implemented as a *MicroBlaze* based microprocessor system utilizing the *Xilinx* Embedded Development Kit (EDK) tool. Further, it is synthesized to determine the number of registers and LUTs consumed on the target platform. For other blocks, i.e., update algorithm and the updatable cryptographic engines, individual hardware descriptions in VHDL are written, simulated, and synthesized utilizing the *Xilinx* Integrated Software Environment (ISE) tool for their evaluation.

4.10.1 *Static Logic Implementation*

The major components inside the static logic are the *MicroBlaze* based microcontroller system and the update algorithm with configuration port. Further, it includes the controller interfaces for UART, external Flash, external memory, and an IP interface (*IPIF*) to the updatable cryptographic engines block in the dynamic region. The core component of the update algorithm being the AES algorithm, figures illustrating its opera-

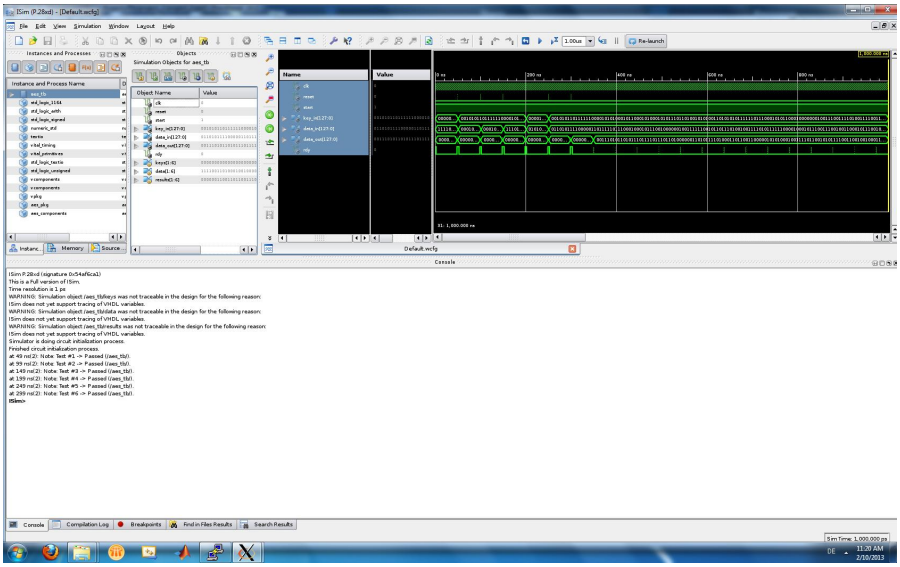

Figure 4.22: Simulation Results of the AES Algorithm

tional schematic, simulation results, and floorplanning on the device are presented in here.

The AES is a symmetric block cipher with a block size of 128 bits and the key can be 128/192/256 bits. In this thesis a 128 bit symmetric key with 10 rounds of operations is utilized [38]. The algorithm utilizes a substitution and permutation type of network (c.f. Figure 4.21), with the main loop of the algorithm involving the operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The first three operations are designed to thwart cryptanalysis via the methods of substitution and permutation, while the fourth operation actually encrypts the data. The plaintext is formatted into 16 byte (128-bit) blocks, and each block is treated as a 4x4 array. Then the algorithm performs aforementioned four operations in each round, except for the last round in which the MixColumns operation is excluded, to generate the output cipher text [38].

The time for update execution is the time amount required for loading, decrypting, integrity & authenticity verifying, and reconfiguring the partial bitstream, i.e., the new asymmetric cryptographic engine or the new hash engine. In our case, it is the AES core that represents the central component of the update algorithm, thus it determines the achievable speed while updating the architecture. The simulation results, as depicted in Figure 4.22, show that the time for 128 bit encryption, decryption, and authentication is 46 ns for each operation. These re-

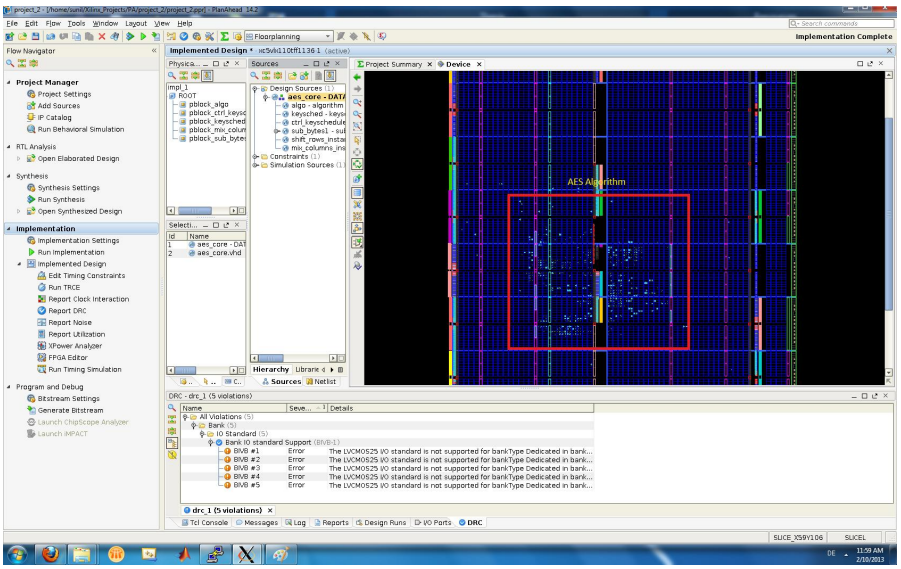

Figure 4.23: FPGA Editor View of the AES Implementation

sults are obtained when the algorithm is executed on a Virtex-5 LX110T FPGA board with *Xilinx* Isim simulator tool. Given this, if the size of the update data is 3 Mb (typical bit stream size in *Xilinx* FPGAs is from 3 Mb to 30 Mb), then the authentication and decryption operations together result in around 2 ms when operating the FPGA at a frequency of 358 MHz. Further, with the configuration port i.e., the Virtex-5 FPGA ICAP supporting a transfer rate of around 3 Gb/s when operating at 100 MHz frequency, would take only 1 ms to configure 3 Mb of data into the dynamic logic of the STPM. Thus, the overall update time is only few milli seconds when loading a new cryptographic engine onto the STPM but one still has to consider the transmission time of update data from the update server in a real application.

The distribution of the resources consumed by the AES algorithm on the device logic is depicted in Figure 4.23. With the available AES implementation in the background, the update algorithm, comprising of AES based cryptographic algorithms, ICAP, and the controller for the overall operation is implemented. The resource consumption values corresponding to the implemented update algorithm are listed in Table 4.1 and additionally an implementation view of the same is depicted in Figure 4.24. The rectangular area (red box) with dots depicted in here is much denser than the one in Figure 4.23, which repre-

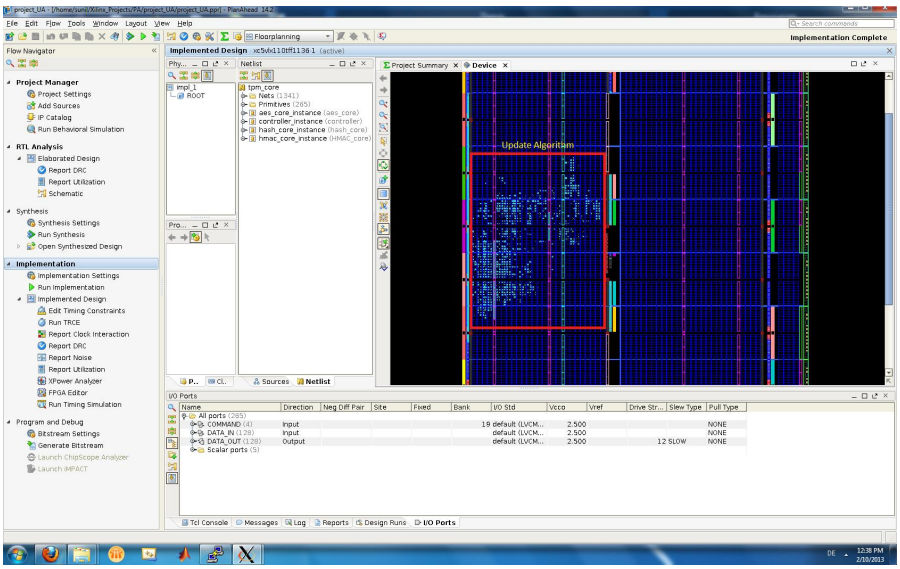

Figure 4.24: FPGA Editor View of Update Algorithm Implementation

Module	Regs.	LUTs	36Kbit BRAM
AES-128 bit	524	899	5
AES-Hash-Core	289	138	0
AES-HMAC-Core	294	184	0
PR ICAP	170	168	2
Controller	662	453	0
Update Algorithm	1939	1842	7

Table 4.1: Resource Consumption of Update Algorithm

sents a higher resource consumption by the update algorithm than the single AES algorithm.

The other major component of the static logic is the execution engine, which in this case is the *MicroBlaze* based microcontroller system. The implementation of the execution engine is split into two parts i.e., the processor itself and the controller interfaces (for accessing the external memories NVM & Flash and the LPC bus). The former part is labeled under *MicroBlaze* and the latter are categorized under peripherals. The overall resource consumption of the execution engine is listed in Table 4.2 and the corresponding editor view is depicted in Figure 4.25.

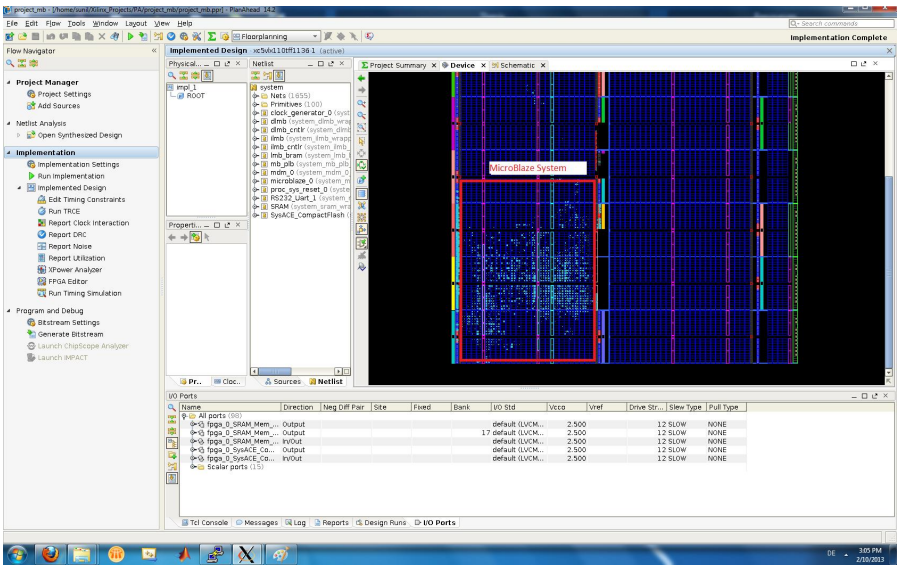

Figure 4.25: FPGA Editor View of Execution Engine Implementation

Module	Regs.	LUTs	36Kbit BRAM
MicroBlaze Processor	2326	2704	4
Peripherals	4629	3351	11
Execution Engine	6955	6055	15

Table 4.2: Resource Consumption of Execution Engine

Table 4.3 summarizes the overall resource consumption of the static logic implementation, i.e., the update algorithm and the execution engine after a synthesis run. It can be seen that the total amount of resources consumed is only 12% of available registers (Regs.) and 11% of available LUTs on the considered FPGA target.

4.10.2 *Dynamic Logic Implementation*

The dynamic logic of FPGA inside the STPM comprises of the cryptographic engines, which when compromised may be replaced with the new cryptographic engines. The resource consumption of these cryptographic engine implementations after a synthesis run is listed in Table 4.4. These engines are loaded into the dynamic logic partition of the FPGA during the STPM operation. It can be seen that the dynamic region may occupy

Module	Regs.	LUTs	36Kbit BRAM
Update Algorithm	1939	1842	7
Execution Engine	6955	6055	15
Total Static Logic	8894	7897	22

Table 4.3: Resource Consumption of STPM Static Logic

Engine	Register	LUT	36Kbit BRAM	Max. Freq. (MHz)
RSA (Mod. Expo.)	7371	8130	7	201.423
ECC (Pt. Mult.)	1207	2455	2	194.476
SHA-1	1013	1754	0	156.14
HMAC	1722	2353	0	156.14
SHA-2	2694	4793	0	271.444
RNG	1424	1248	5	283.68
Cryptographic Engines	15431	20733	14	

Table 4.4: Resource Consumption of STPM Cryptographic Engines

the major portion of the FPGA because the static logic is rather small in size. Therefore, the new cryptographic engines to be loaded onto the device find plenty of reconfigurable resources.

Though the update algorithm inside the static logic comprises of a hash engine, i.e., AES-Hash, the dynamic logic is loaded with a new SHA-2 engine, in case of a compromise, because most of today's computers utilize only the standard SHA series algorithms. Further, this choice is made to comply to the standard TCG specifications stating the use of these hash algorithms in current and future TPM technology. For an asymmetric algorithm, to evaluate the efficiency of its implementation, it is typical to implement and evaluate its fundamental operation. In this context, for evaluating RSA and ECC, the modular exponentiation operation and the point multiplication operation respectively are implemented on the target platform. It can be seen that the latter operation consumes significantly less resources compared to the former. Additionally, the sum of resources consumed by ECC, SHA-1, HMAC, SHA-2, and RNG is almost equal to

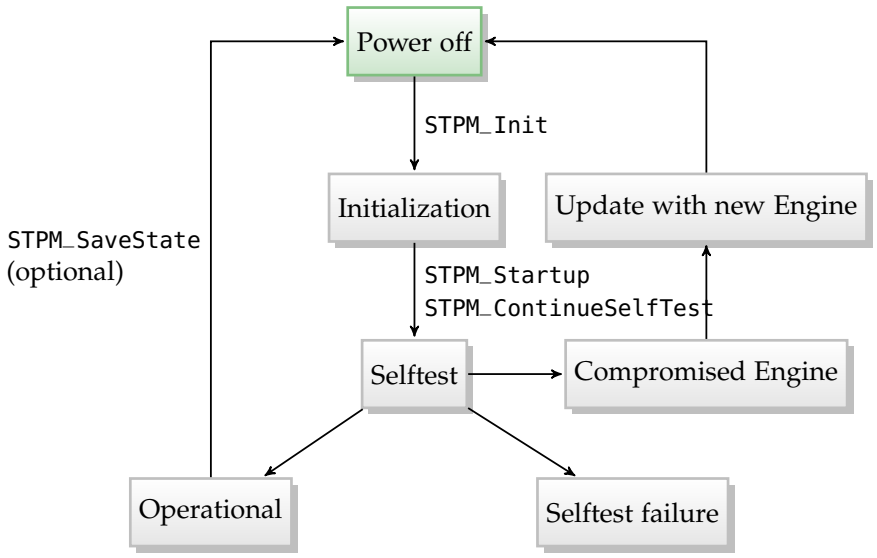

Figure 4.26: STPM Life Cycle

that of resources consumed by the RSA alone. Therefore, the replacement of the RSA engine by the ECC engine is justifiable.

With reference to Table 4.3 and Table 4.4, the current STPM design implementation occupies around 35% of available registers and 41% of available LUTs on the considered FPGA target. In addition to the static and dynamic logic resources, the FPGA of the STPM is attached with an external Flash memory of size 1 GB, which is large enough to hold both the platform keys and the bitfiles, thus providing the combined features of NVM and programmable Flash for the STPM design. This external Flash is accessed by the *MicroBlaze* processor over a dedicated Flash controller, referred to as *System ACE CF Controller* [102]. Furthermore, any additional requirements such as tamper detection and others may be included in the design as and where required.

The flexibility of loading different cryptographic engines on the STPM architecture provides the system with a level of security, which is even higher than the security provided by the conventional TPM. This is because, for example, the newly loaded cryptographic engines onto the STPM may feature side-channel or quantum crypto resistance properties. Further, the Flash memory may hold various implementations of the required cryptographic algorithms such that the user can choose from. The aforementioned two features satisfy the requirements of the *TPM.Next* specification by the TCG.

The life cycle of the STPM, as depicted in Figure 4.26, is similar to the life cycle of conventional TPM but with an additional capability to handle the compromised cryptographic engines (c.f. Figure 2.2). As an example application, the TPM plays a major role in an enterprise security by providing a number of trusted cryptographic and information security capabilities. One such capability is the trusted boot, which in contrast to the secure boot, takes only measurements upto the boot loader and leaves it upto the remote party to determine the system's trustworthiness. This and the other capabilities such as user authentication and data protection rely on a secure RSA and SHA-1 algorithms whose compromise leads to a loss of huge enterprise data. Thus, utilizing the novel architecture proposed in this work, i.e., the STPM, the compromised hash and asymmetric algorithms may be updated with new correspondingly uncompromised algorithms to protect sensitive enterprise data.

5

APPLICATIONS OF STPM FEATURES

The PR capability of the underlying FPGA is one important feature that made the design of the STPM possible. There exist some more advantages of the PR technology that may be used in various applications. Similarly, one more significant component of the STPM, i.e., the update algorithm, whose capabilities may be applied in other domains such as automotive electronics, in specific to real-time embedded systems. With reference to above, two different applications that utilize the PR feature and the update algorithm of the STPM respectively are illustrated in this chapter.

5.1 A NOVEL IP PROTECTION SCHEME

Currently SRAM based FPGAs are becoming increasingly popular as building blocks of electronic systems because of advantages such as easy design modification (reconfigurability), rapid prototyping, economical cost for low volume production, and availability of sophisticated design and debugging tools. Applications of FPGAs in the area of consumer electronics include, for example, television circuits, communication and video processing devices, and software-defined radios. Considering that the FPGAs are becoming so important for the electronic industry, it is necessary to think about the security of FPGA based systems. The security of FPGA requires two fundamental measures to be taken i.e., protecting the FPGA data and the FPGA design itself. In the former case, it is necessary to protect the FPGA application i.e., the data inside the circuit and the data transferred to/from the peripheral circuits during the communication. Whereas in the latter, the concerns are against cloning and reverse engineering the FPGA design i.e., the configuration data, which is the IP of the original designer. Thus, with the increase in deployment of FPGAs in modern embedded systems, the IP protection has become a necessary requirement for many IP vendors, as presented in sequel.

There exist many proposals in the literature for IP protection, from commercial vendors such as *Xilinx* and *Altera*, using symmetric encryption techniques. However, these methods have a

drawback i.e., they need a cryptographic key (i.e., the symmetric key) to be stored in a non-volatile memory located on FPGA or in a battery-backed RAM as done in some of the current FPGAs. The expenses with such methods are, occupation of larger area on FPGA in the former case and limited lifetime of the device in the latter, the requirements which are crucial for resource constrained embedded systems. In contrast, a novel method which combines the dynamic PR feature of an SRAM based FPGA (c.f. 4.2) with the PKC (detailed later) to protect the FPGA configuration files without the need of fixed key storage on FPGA or external to FPGA, is proposed in here [63]. Further, the proposed scheme is implemented as a proof-of-concept on *Xilinx* Virtex-5 FPGA platform to illustrate the feasibility of the design.

A brief description of various possible attacks on the FPGAs is given in the following:

- *Cloning* is when a competitor makes a copy of the design, and when he is able to make a copy of the pirated system. With FPGAs it is very simple to clone an unprotected design as the bitstream can be copied to another FPGA's configuration memory.

- In the case of *reverse engineering* the design is copied by reconstructing a schematic or netlist level representation. As demonstrated in [73], methods intended to convert FPGA bitstreams into netlists are relatively easy to apply and may soon deliver widely usable results. The *reverse engineering* is more serious than *cloning* because in the former, the attacker understands how the design works and may modify it with malicious intent. These two correspond to different attacks, and the design security must protect the system against both these attacks. The papers [13] and [14] give some information about these different attacks.

- The *non-invasive* attacks gather all the methods that use external means. For example the attackers may use all the possibilities of the circuit inputs in order to obtain all the different outputs and draw the system truth table, this method is called "black box attack". In the case of an SRAM based FPGA, a simple attack method can be, intercepting the bitstream between the root ROM and the FPGA when the power is switched on. More complex attacks can use power and electromagnetic changes and measures like the simple or differential power analysis as presented in [66].

- The *invasive* (- or *physical*) attacks are characterized by the necessity to destroy the integrated circuit (component package) to study the chip (design inside the component) using some complex methods. For example, it is possible to use a laser cutter microscope in order to split the chip in several slices and understand the chip layout as discussed in [24]. These attacks may use sophisticated tools such as optical microscope and mechanical probes. As these attacks use the weakness of the silicon technology, when they are possible, it is very hard to secure the system against them.

However, the proposed method, is secure against only the known attacks such as the Man-In-The-Middle (MITM) attack and the replay attack, which are categorized under *non-invasive* attacks. Especially, the method protects the bitstream against interception by an attacker on the communication channel (Internet/Non-Internet based) between IP vendor (local or remote) and system developer (local or remote). Furthermore, using this novel method not only high-end FPGAs but also low-end FPGAs with PR capabilities are secured. One application scenario for the proposed method can be a secure IP deployment in an embedded system design in the automotive electronics.

The PKC scheme utilizes asymmetric key algorithms such as RSA and ECC, which unlike symmetric key algorithms, do not require a secure initial exchange of key between the sender and the receiver. The asymmetric key algorithms are used to create a mathematically related key pair: a secret private key and a published public key. Messages are encrypted with the recipient's public key and can only be decrypted with the corresponding private key, which is known only to the receiver. In Figure 5.1, we see that all of the functions in the "dashed box" (named as secure box) may be implemented within the physical package of the FPGA. The plaintext and the private key information never leave a well-protected container i.e., the secure box.

5.1.1 *Conventional Schemes*

There exist some solutions in the literature to overcome the key storage and additional battery problems for the IP protection, as described next. Tom Kean of the *Algotronix* society proposed ideas to store the cryptographic secret key on FPGA, such as using laser to program a set of links during manufacture [55].

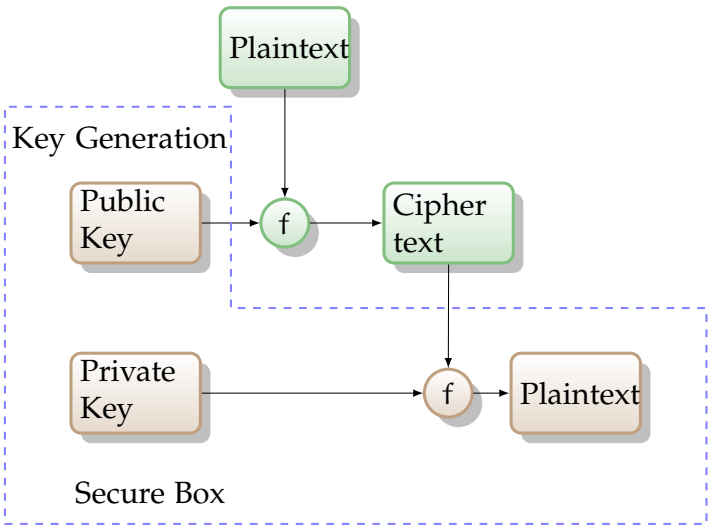

Figure 5.1: Public Key Cryptography

However, in his method the encryption and decryption circuits are embedded inside the FPGA which causes less available silicon area for developed applications. Furthermore, the encryption and decryption circuits are fixed, thus it is not possible to upgrade them. In contrast, Yip et al. have proposed an IP protection scheme that utilizes partial-encryption technique [109]. The authors in here claim that their technique outperforms the full-encryption technique in terms of the reverse engineering cost. Guajardo et al. have proposed a different scheme that utilizes FPGA's intrinsic PUF and asymmetric cryptography [47]. Though their method uses asymmetric key based authentication protocol which does not need the private key to be stored on the FPGA, they did not make use of the advantages provided by the PR feature. Additionally, the PUF implementation and its analysis on the FPGA is in itself a challenging task.

The other technique is the watermarking proposed by Lach et al. [61], where they apply a watermark to the physical layout of a digital circuit. This watermark when onto an FPGA, uniquely identifies the circuit origin and is yet difficult to detect. In contrast, Güneysu et al. used both public key and symmetric key cryptography to dynamically protect the IP of circuits in configuration files [48]. In their method the symmetric cryptography is hard-wired and the public-key functionality is moved into a temporary configuration bitstream for a one-time setup procedure. Also, Bossuet et al. have proposed a scheme where an embedded key is accessible to the user logic and uses PR to encrypt and decrypt the bitstream [21]. Here, an on-chip key is used to

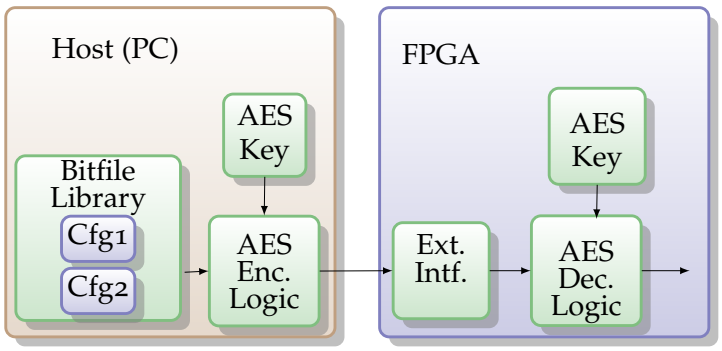

Figure 5.2: Conventional Scheme for IP Protection

encrypt the main design's bitstream before storing it in a programmable ROM where a decryption bitstream is also stored. In [82], Simpson et al. have proposed an off-line authentication scheme for secure delivery of IP modules in non-networked environments of FPGA design. Their scheme implements a mutual authentication of the IP modules and the hardware platform, thereby enabling the authentication and integrity assurances to both system developer and IP provider.

However, all these methods need a secret key to be stored on an FPGA which in itself is a challenge in SRAM based FPGAs as the memory on these devices is volatile. In contrast, the novelty of our method is that it does not need any fixed key storage, either on the FPGA or outside of it because the keys are generated on the fly. There is no threat of the private key being stolen, as it is stored (temporarily) deep inside the memory blocks which are erased when the device is turned off. Furthermore, the keys are generated on the FPGA (simultaneously on the IP vendor side) which may be used for secure deployment of IP in the field.

The conventional IP protection scheme utilizing the symmetric encryption is depicted in Figure 5.2. The Host (PC) first encrypts the bitstream with a symmetric encryption algorithm such as AES using a secret key before sending it to the FPGA. Later, on the FPGA side, the bitstream is decrypted using the same stored secret key and the corresponding decryption unit.

5.1.2 *Proposed Novel Scheme*

The proposed novel scheme is depicted in Figure 5.3. The considered FPGA supports the dynamic PR, and is divided into static and dynamic logic regions. The initial bitfile (full bit-

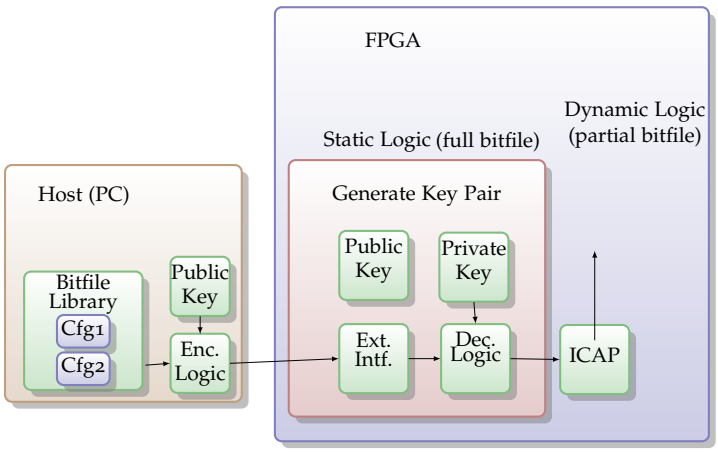

Figure 5.3: Novel Scheme for IP Protection

stream) to be loaded into the static part of the FPGA is an unencrypted design that does not feature any proprietary information. It only contains the algorithm to generate the public-private key pair and the interface between the host, FPGA, and the ICAP. The individual steps of the scheme are illustrated in Figure 5.4.

However, there are Flash-based FPGAs available in the market, such as *ProASIC3* device from *Microsemi*, over the SRAM based FPGAs for key storage but the former are not in common use yet. The proposed system works well when we consider an in-house scenario i.e., the loading of the IP from a PC onto the FPGA connected directly to it. But when the IP is to be loaded from a remote location or over Internet then there is a possibility of an attack on the system such as the MITM attack. In MITM attack, an adversary tries to intercept the communication channel between the vendor and the user and he can deceive the system in two possible ways. In one case, he can pose himself as the user and send his own generated public key to the vendor for encryption inorder to decrypt it later, with his private key, which is the IP theft scenario. In the other case, he can pose himself as the vendor to receive the public key from the user and send back encrypted malicious configuration, which on after loading on to the system would destroy the device. These two cases clearly show that there is a lack of mutual authentication for data origin in the system.

One solution to avoid the MITM attack is by providing an additional, non-Internet based channel for transmission of the public key from user to the vendor. The user may exploit the Universal Mobile Telecommunications System (UMTS) stick for

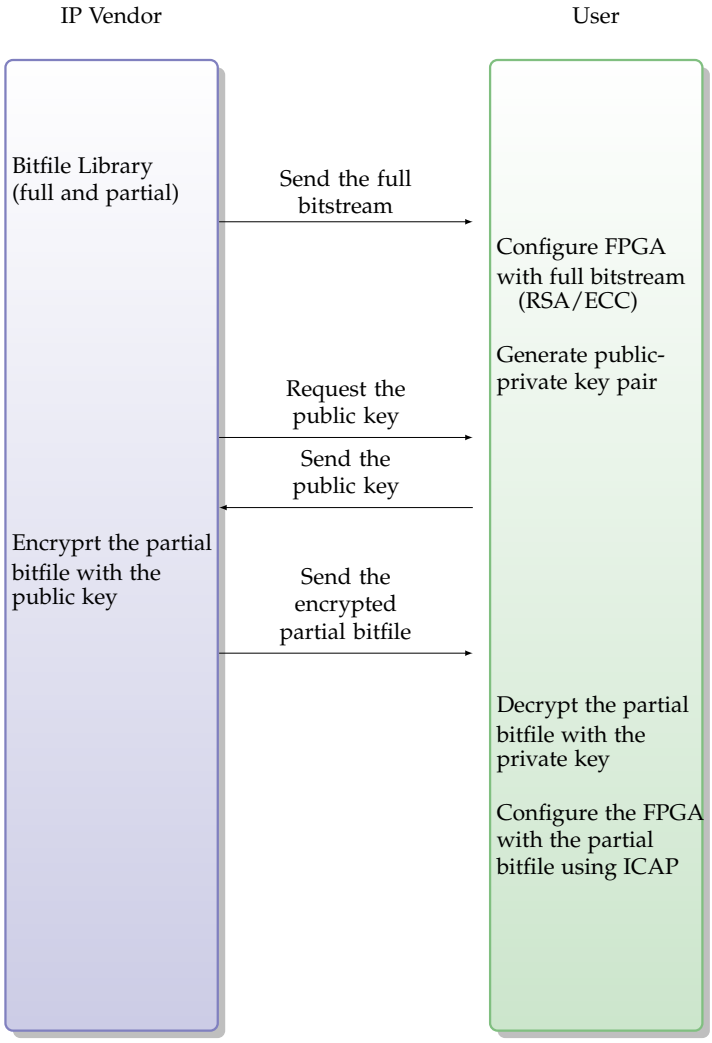

Figure 5.4: Protocol for Secure IP Deployment

a mobile phone link to the configuration files server of the vendor. Thereby, only the vendor and the user would know the public key being transmitted over the channel, which guarantees the authenticity of the IP origin. However, there exist other solutions available in the literature for thwarting the MITM attack such as the MAC based schemes and signature schemes.

Following are the advantages of the proposed scheme:

- The public-private key pair may be regenerated at any time. If a new configuration is downloaded from the host it may be encrypted with a different public key and decrypted with the corresponding new private key. Even if the FPGA is configured with the same partial bitfile at a later instance, such as after a power-on-reset, a different public/private key pair may be used.

- There is no need of any non-volatile memory to store the key (as for symmetric key in previously mentioned scheme) for decrypting the bitfiles as the private key is generated on the fly. In addition, the private key generated by the asymmetric algorithm running on the FPGA is stored in the SRAM and if the FPGA loses power then the private key no longer exists as the SRAM memory is volatile.
- In general, the partial bitfile contains the vast majority of the FPGA design with the logic in the static design consuming a very small percentage of the overall FPGA resources. Thus, most of the FPGA resources are allocated to the actual applications contained in the partial bitstreams.
- Even if at some point of time it is found that the asymmetric algorithm being used is no more secure, one can replace it with a new algorithm as it just requires loading a new full bitstream into the static region of the FPGA.
- The issue of loading IPs from different vendors onto a single SoC may be addressed.

There are also certain disadvantages with this scheme, such as the implementation of asymmetric key algorithm (RSA) on the FPGA consumes a lot of resources. However, this can be avoided by replacing with efficient algorithms such as ECC. Although the generation of partial and full bitstreams for the FPGA is cost expensive, time expensive, and exhausting at the moment, the FPGA vendors claim that it will be much easier on top of their newer versions of tools.

5.2 SECURE REAL-TIME SYSTEMS

There is a continuous rise in the deployment of electronics in today's systems especially in automobiles, thus the task of securing them against various attacks is becoming a major challenge. In particular, the most vulnerable points are: communication paths between the Electronic Control Units (ECUs) and between *Sensors* & *Actuators* and the ECU, remote software updates from the manufacturer and the in-field system. However, when including additional mechanisms to secure such systems, especially real-time systems, there will be a major impact on the real-time properties, such as deadlines and schedulability, and on the overall performance of the system. Thus to analyze the

impact on the aforementioned properties, a minimal security module is deployed in a target real-time system, while achieving the goals of secure communication and authentic system update.

Real-time applications such as railway signaling control and car-to-car communication are becoming increasingly important. However, such systems require high quality of security to assure the confidentiality and integrity of the information during their operation. For example, in a railway signaling control system, the control center must be provided with data about position and speed of the approaching train so that a command specifying which track to follow may be sent back. In such a case it must be assured that the messages exchanged between the two parties are not intercepted and altered by a malicious entity to avoid possible accidents. Additionally, it is mandatory to confirm that the incoming data to the control center is in fact from the approaching train and not from an adversary. Similar requirements are needed in a car-to-car communication system. Furthermore, above considered systems are highly safety relevant, thus the real-time properties typically play an important role in such systems. In general, the goal of a real-time system is to satisfy its real-time properties, such as meeting deadlines, in addition to guaranteeing functional correctness. This raises the question, what will be the impact on these properties of such a system when including, for example, security as an additional feature?

There exist some work in literature which addresses the issue of including security mechanisms inside real-time applications. For example, Lin et al. [62] have extended the real-time schedulability algorithm Earliest Deadline First (EDF) with security awareness features to achieve a static schedulability driven security optimization in a real-time system. For this, they extended the EDF algorithm with a group-based security model to optimize the combined security value of selected security services while guaranteeing schedulability of the real-time tasks. In a group-based security model, security services are partitioned into several groups depending on the security type and their individual quality so that a combination of both results in a better quality of security. However, this approach had a major challenge as how to define a quality value for a certain security service and to compute the overhead due to those services. In another work, authors Marko et al. have designed and implemented a vehicular security module, which provides trusted

computing like features in a car [99]. This security module protects the in-vehicle ECUs and the communication between them, and is designed for a specific use in e-safety applications such as emergency breaking and emergency call. Further, the authors have given technical details about hardware design and prototypical implementation of the security module in addition to comparing its performance with existing similar security modules in the market. Additionally, the automotive industry consortium, autosar, specified a service, referred to as Crypto Service Manager (CSM), which provides a cryptographic functionality in an automobile, based on a software library or hardware module. Though this service is based on a software library, it may be supported by cryptographic algorithms at the hardware level for securing the applications executing on the application layer. However, none of the aforementioned approaches addresses the issue of analyzing the impact on the real-time properties of a system when integrating a hardware security module inside it. In the following, a description of the target system under consideration is given before evaluating the impact of including security features inside it.

5.2.1 *Target System Specification*

The system under consideration is depicted in Figure 5.5. It comprises of a *Sensor*, an *Actuator*, an ECU with a processor & a security module, and an update server. The system realizes a simple real-time control application, where *Sensor* data are processed by the control application in order to operate the plant due to an *Actuator*. However, the concrete control application is not of interest in this context. It might represent the engine control of a car, or a driver assistant system such as an Automatic Breaking System (ABS).

The scenario depicted in Figure 5.5 consists of the following flow: *Sensor* periodically delivers data from the plant over the bus (1), which is in an encrypted form to avoid its interception and cloning by an attacker. The data is received by the input communication task *ComIn*, which is part of the operating system (OS). Each time the input communication task receives a packet, it calls the security service (*SecSrv*), which is also part of the OS, for decryption of the packet (2). The security service provides the hardware abstraction for security operations, and schedules service calls. The decryption call from the communication task is forwarded to the security module (3), which

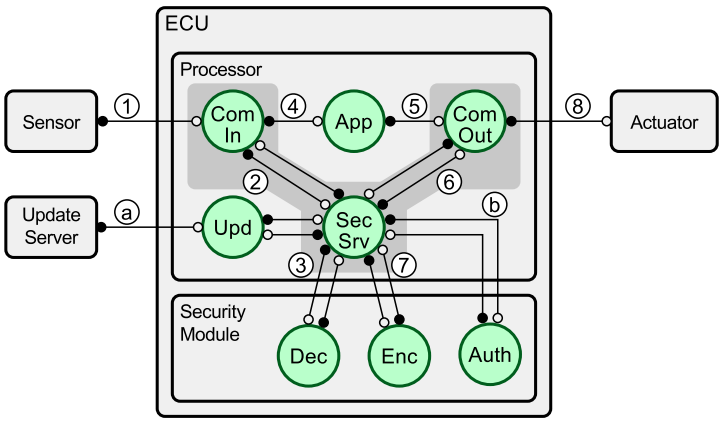

Figure 5.5: Target System Scenario

processes the packet data. The cryptographic operations of the security module modeled by *Dec*, *Enc*, and *Auth* are realized as hardware blocks. The decrypted data is sent back to the security service, which is in turn returned to the task *ComIn*. Now the data is ready for transmission to the application (4), which is modeled by a single task *App*. The application task is activated by the incoming packet, and processes the *Sensor* data. The controller implementation of the task calculates respective *Actuator* data and sends it to the communication task *ComOut* (5) for transmission to the *Actuator*. However, before sending the data to the *Actuator*, the communication task again calls the security service (6), which in turn accesses the security module for data encryption (7). After *Enc* has encrypted the data, it is sent back to the communication task *ComOut* via *SecSrv*, which delivers the packet to the *Actuator* (8). It is required that the control application finishes the described flow within a single control period, i.e., before the next *Sensor* data arrives. Additionally, the system implements a function for software updates. To update the system with new software, the *UpdateServer* sends the data to the *Upd* task (a) via a communication medium (e.g., Internet) to the outside. This received data must be authenticity verified and decrypted before loading it into the system. For this, the task *Upd* forwards the data to the *SecSrv*, which utilizes the *Auth* block of the security module (b). Only after a successful authentication, the data is decrypted and loaded into the system else it is rejected.

The security module, integrated into the ECU, is a hardware module of the STPM (c.f. Section 3.5.2), comprising of cryptographic hardware blocks for performing operations such as encryption, decryption, and authenticity verification. Though

these operations are denoted as tasks in the system view, they are implemented as hardware blocks. Further, a controller (a state machine), a memory block, and an I/O block are included inside the security module (though not depicted in Figure 5.5). Whereas the controller executes the commands for the aforementioned cryptographic operations, the memory block acts as a data buffer, and the I/O is an interface to the processor of the ECU. The commands arrive as requests form the *SecSrv* on the processor, and the responses from the security module are sent back. In essence, the *SecSrv* acts as a software abstraction of the hardware security module, for providing required cryptographic operations to the executing tasks on the processor. The security module is equipped with particular support for update functionality i.e., authenticity verification. In normal operation, the data is temporarily stored in the memory block of the security module to compare the attached MAC value by the update server with the computed MAC value in the security module before decrypting and loading it. This kind of operation, where all update data is stored in memory of the security module, and authentication being applied at once on the data, is however not appropriate in the context of the considered real-time application. This is because, while the security module is performing authentication, other operations such as decryption of incoming *Sensor* data or encryption of outgoing *Actuator* data are blocked. Given this, large update data can block the device for a long time span, resulting in a violation of allowed delay by the control algorithm.

To avoid such a situation in the considered scenario, for authenticity verification, the MAC is calculated in two steps. First a checksum of the update data is calculated using a public hash algorithm such as SHA-1 and then a MAC is computed on this checksum. The *Upd* task thus calculates and sends only the checksum of the data instead of whole data itself to the security module via *SecSrv*. Considering the update process being not time critical, *Upd* task is executed with low priority, preventing any undesired interference with the real-time application. Therefore, only the interference for authentication of the single checksum has to be considered. However, since the update data being encrypted, *Upd* task needs to access the security module for its decryption. To this end, the data is split into packets and decrypted piece-wise though the impact of these operations has to be considered in the real-time analysis.

Another possibility to verify the authenticity of the update data would be to compute the MAC iteratively within the security module. For this, the update data from the *Upd* task is sent to the security module in a block-by-block basis via the *SecSrv*. The computed HMAC on the received block is stored inside the memory block of the security module. Before sending the next block, the *SecSrv* checks for any pending requests for encryption or decryption operation from other high priority tasks. If there exist such a request, it is executed before sending the next block of data for MAC computation. In order to handle this procedure, the security module should be equipped with an additional hardware block performing the scheduling of cryptographic operations. Further, the communication interface between the *SecSrv* and the security module has to be modified. Though, this method is currently not supported by current security module design, it is definitely a desired feature.

Though it is not depicted in Figure 5.5, the *Sensor* and the *Actuator* must also be equipped with the ability to perform encryption and decryption operations respectively. This is because, at an higher level view of the system, the *Sensor* after obtaining the data from the plant has to encrypt it before transmitting it over the bus and the *Actuator* has to decrypt the received data from the ECU before providing it to the plant. To perform these operations they must also be equipped with the encryption and decryption blocks, along with the corresponding secret key as in the security module. For this, an integrated non-volatile memory may be utilized to store the secret key, which is programmed into *Sensor*, security module, and *Actuator* by system manufacturer during the system deployment. Additionally, the security module must hold an additional secret key for performing authentication operation on the update data. The features of the security module here are identical to the ones in hardware module of the update algorithm (c.f. Section 3.5.2) i.e., the former utilizes the same cryptographic blocks as the latter. Thus, the security module provides a secure communication path between the *Sensor* and the *Actuator* and to provide authentic updates of the system. Additionally it consumes very few computational resources, which is essential in resource constrained embedded systems.

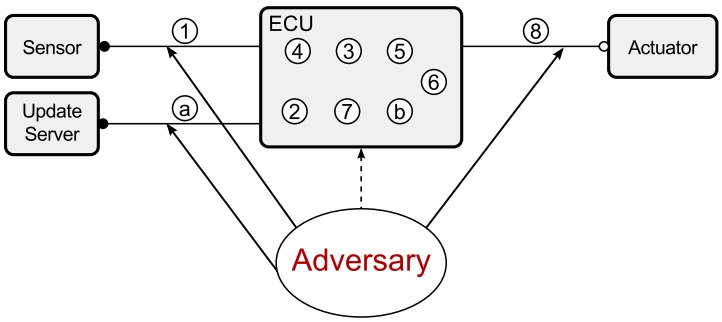

Figure 5.6: Considered Adversarial Model

5.2.2 *Adversarial Model*

To describe all possible attack points in the considered system, an adversarial model is formulated as depicted in Figure 5.6. The model highlights all the components (with a simplified ECU block) and the corresponding internal and external communication paths (i.e., numbered circles) of the original system (c.f. Figure 5.5). The adversary considered in the model is an active eavesdropper (c.f. Dolev-Yao Model [34]) and is able to perform attacks as described in taxonomy of cryptographic attacks by Popp in [75]. For an analysis here, it assumed that the attacker is only able to perform classical cryptanalytic attacks on the external communication links (indicated by thick arrows coming from adversary) i.e., from *Sensor* to ECU, ECU to *Actuator*, and update server to ECU. The details of classical cryptanalytic attacks and techniques to overcome them are already given in Section 3.3. With reference to this, the security module of the target system is embedded with cryptographic blocks providing such techniques. We rule out the possibility of attacker being eavesdropping the ECU's internal communication (indicated by a dotted arrow coming from adversary) because such an attack implies that the attacker is having a physical access to the ECU and thus control the running OS and the tasks themselves. Further, we do not consider any Denial-of-Service (DoS) attacks by the attacker on any of the components of the considered system.

5.2.3 *System Analysis*

To analyze the impact of including the security module on the real-time properties of the system, we consider three different test cases as detailed in the sequel. A brief description about

the system set-up and the utilized tools is given before delving into the obtained results with the test cases.

The control application is executed with a frequency of 10 kHz, i.e., the *Sensor* sends each 100 µs a data packet to the ECU. The update service is modeled as a sporadic application with typically very large time spans between individual invocations. All tasks of the processor are scheduled by a fixed priority scheduling scheme with preemption, where lower priority tasks can be interrupted by higher priority tasks. Furthermore, all tasks belonging to the OS (depicted by a dark gray shaded area of Figure 5.5) get higher priority than the application tasks. The priorities in descending order are *ComIn*, *ComOut*, *SecSrv*, *App*, and *Upd*. The operations of the security module are not scheduled, and the module can be considered as a shared resource. The *SecSrv* task processes incoming security operation requests for the security module in First-In-First-Out (FIFO) order.

For all test cases, the processor of the ECU is a 50 MHz processor (20 ns cycle time) that is equipped with internal memory for storing data and code. Internal memory is accessed by reading and writing 16 bit words within a single processor cycle. Communication between the ECU, the *Sensor*, and the *Actuator* is realized by a Controller Area Network (CAN) bus. For simplicity, it is assumed that all data are transferred between the processor and the CAN bus interface via I/O registers of 16 bit width, and with a delay of four processor cycles. Communication over CAN is restricted to 64 bit user data, and is assumed that this is also the size of packets transmitted between the ECU and the *Sensor*/*Actuator*. In order to transmit 128 bit data as required by the operation of the security module, each transmission consists of two packets. Receiving and transmitting data thus requires 16 Byte data transfer between the CAN bus controller and the processor, summing up to 64 processor cycles (1.28 µs). Storing the packet into the OS internal memory costs additional 320 ns. Bus latencies are not further specified in this setting, as the concentration is mainly on the timing of the ECU application.

The utilized AES algorithm inside the security module operates on 128 bit blocks of input data at a time. Thus, all the blocks (*enc*, *dec*, and *Auth*) of the security module, operate on same data size because they utilize the same algorithm. The security module is implemented as a proof-of-concept on a *Xilinx* Virtex-5 FPGA [108] platform. The individual cryptographic

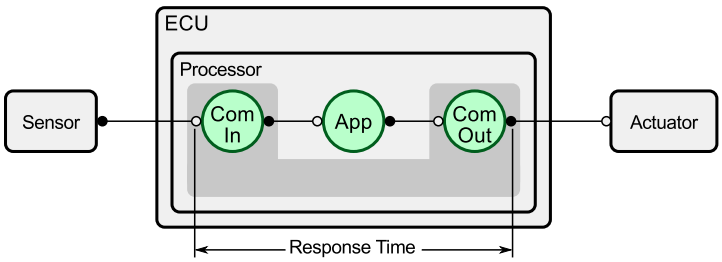

Figure 5.7: Scenario 1 w/o security feature

blocks of the security module are simulated and synthesized utilizing the device specific tools. Utilizing an operating frequency of 358 MHz for the FPGA, the execution time for each of encryption, decryption, and authentication operations is determined (by simulation) to be 46 ns for a 128 bit block of input data. The timing parameters for other operating frequencies of the security module are obtained by simple scaling. The utilized FPGA device supports storage in the form of BRAM with 36 kb size, which is large enough to be used as memory block of the security module.

The timing analysis is applied in order to find the worst-case end-to-end response time of the control application, starting from the reception of *Sensor* data up to the sending of *Actuator* data (as depicted in Figure 5.7). Various static scheduling analysis tools are available for this analysis (e.g., [50]). The system however is sufficiently small for a more precise analysis based on real-time model-checking [32]. To this end, the system is translated into a Uppaal model [18]. The worst-case response time is obtained by a binary search on the value range of respective model variable.

5.2.3.1 *Target system without any security features*

In the first scenario, which is depicted in Figure 5.7, the communication tasks send the data directly to the control application and to the communication bus without encryption or decryption. The resulting end-to-end response time is shown in column "Scenario 1" of Table 5.1. As expected, the analysis shows that the execution times are simply summed up for the involved tasks since no further interferences occur in this simple setting. Indeed the situation would be different when multiple application tasks are executed on the same ECU, which would cause additional interferences.

Task	Scenario 1	Scenario 2	Scenario 3		
		50MHz LPC	50MHz LPC	50MHz MMIO	358MHz MMIO
App	50.0 us	50.0 us	50.0 us	50.0 us	50.0 us
ComIn	1.6 us	1.6 us	1.6 us	1.6 us	1.6 us
ComOut	1.6 us	1.6 us	1.6 us	1.6 us	1.6 us
SecSrv	—	80 ns	80 ns	80 ns	80 ns
Comm. CPU/SM	—	1.46 us	1.46 us	400 ns	200 ns
Dec	—	358 ns	358 ns	358 ns	46 ns
Enc	—	358 ns	358 ns	358 ns	46 ns
Auth	—	—	358 ns	358 ns	46 ns
Response Time	53.2 us	60.1 us	62.0 us	56.7 us	54.96 us

Table 5.1: Analysis Results

5.2.3.2 *Target system with secure communication feature*

In the second scenario, only the secure communication feature between the ECU and the *Sensor* and the *Actuator* is enabled but the update service is switched off. This implies that the security module has to perform only encryption and decryption but no authentication. For this scenario, we assume that the security module communicates with the processor via the LPC bus [51]. According to the specification [51], the transfer of 128 bit data plus 16 bit command requires about 1.46 us, when the bus operates with typical timing parameters. Each invocation of the *SecSrv* involves a transfer of the data to or from the security module, plus the execution time of the task of 80 ns for internal copying operations. The security module operating at a clock rate of 50 MHz results in an individual execution time of 358 ns for encryption and decryption (from simulation results). With this set-up, the timing analysis shows (column "Scenario 2" of Table 5.1), that enabling only the secure communication feature results in a significant raise in the response time of the system i.e., about 13% more than in the previous scenario.

5.2.3.3 *Target system with secure communication and secure update features*

For the third scenario, both secure communication and authentic update features are enabled. This scenario has been analyzed with three different sets of timing parameters.

The first setting assumes the same parameters as for "Scenario 2". Hence the security module operates at a clock rate of 50 MHz, and communicates with the processor via LPC bus. The results show a further raise in the end-to-end response time of the application because the update service might call the authentication service, which incurs an additional execution time to the pending encryption and decryption operations of the security module. Thus, it can be seen that the end-to-end response time is around 16% higher than in the first scenario. For the second setting, it is assumed that the security module communicates with the processor via MMIO. Memory transfers are assumed to operate with 16 bit words, and a delay of four processor cycles, resulting in transfer times of 80 ns. A transfer between the processor and the security module now sums up to 400 ns. The final setting works with a very fast security module, and memory transfers only having two cycles delay. The security module operates at 358 MHz, which results in all cryptographic operations with 46 ns of execution time. Surprisingly, the end-to-end response time in the second and the final setting has reduced and is only around 6.5% and 3.5% respectively. This implies that the type of communication interface between the processor and the security module has a significant impact on the resulting overall response time of the system.

In all settings, the software update function is assumed to perform the operations as discussed in Section 5.2.1. After calculating the checksum of the update data, task *Upd* sends an authentication request to *SecSrv*. When the authentication is successful (which is always true in the considered scenario), the task successively sends decryption requests for each packet of the update data, while waiting for the reply before sending a new request. The results shown in Table 5.1 represent the worst-case behavior obtained with various values of the execution time (between 10 ns and 1 us) needed by *Upd* between successive decryption requests. The impact of the operation of *Upd* remains rather small, which can be explained by the fact that the task is executed with low priority. However, the selection of the execution times is not exhaustive, and thus do not guarantee absence of race conditions. To enforce limited impact of

the update function, the *SecSrv* should be modified by running with priority inheritance, where requests are executed with the same priority as the calling task.

To summarize, it is observed that, with the integration of such a security module into the ECU, the response time of the system is strictly dependent on the utilized communication interface between the ECU processor and the security module. In specific, the worst-case response time of the system is high for a slower interface (i.e., LPC) and decreases drastically for a faster interface (i.e., MMIO). Thus, when including a security mechanism in real-time systems it is necessary to consider about the type of communication interface being utilized.

6

CONCLUSION

The audit of the trust units should be used reliably feasible for users. Today's trust anchors, however, are not in a suitable position to guarantee the system integrity in amended new requirements. In this context, an innovative reconfiguration concept allows the adjustment of existing trust anchors, for example for re-establishment of integrity even after a compromise of security algorithms and for providing future-proof customized functionality. The unique feature of this concept is the ability to securely exchange encryption algorithms without modifying the architecture of the trust anchor. This is achieved by the reconfigurability concept and thus avoiding the replacement of complete component itself. The details about such a concept and the resulting design utilizing it, which form the main goal of this work, are summarized below.

A novel architectural concept for updating the cryptographic engines embedded in a TPM is proposed and implemented in this thesis. First, the state-of-the-art technology, i.e., the TC approach and the TPM security functionalities are detailed. Then, a thorough analysis of the possible threats to the cryptographic engines on the TPM and the implications of these threats on its security functions is performed. From this analysis, an investigation is performed to determine the functional and non-functional requirements that arise due to compromised cryptographic engines, in specific RSA and SHA-1, on the TPM. This analysis, referred to as STC, led to defining the architectural requirements of the new updatable TPM design that supports a flexible and secure update of the compromised engines. Utilizing the STC concept, a novel architecture called STPM with required features is proposed.

The STPM architecture comprises of all the required components such as the execution engine, update algorithm, configuration port, memories, and a region for updatable cryptographic engines to provide the features of a conventional TPM. The execution engine is a microcontroller based system, which executes the commands arriving from the processor of the system utilizing the running command software on it. The update algorithm, a hardware module, utilizes a special format for update

data for securely updating the cryptographic engines. In addition to providing an updatability feature, the STPM stays secure against the possible attacks, which were formulated utilizing an adversarial model. Furthermore, the STPM re-establishes the trust after an engine update which is a mandatory requirement in trustworthy systems. A proof-of-concept implementation of the STPM architecture utilizing available components in the market, in specific, on the *Xilinx* Virtex-5 LX110T FPGA platform is detailed. On this implemented architecture two test cases for updating the fundamental engines SHA-1 and RSA are also presented. Two real-world scenarios that may utilize the features of the STPM are illustrated as a part of the applications.

Given that the proposed STC concept is generic, it is upto the manufacturer to decide on the architectural view but it must be guaranteed that all the requirements of the proposed concept are met. Though, software-based solutions are also possible in principle, but the complexity of the cryptographic computation method results in long computation times on the processors, which is not desired. More powerful processors are often not an option, since both the total cost of a system and the energy demand would rise unduly. Further, the utilized platform i.e., the Virtex-5 FPGA, for a proof-of-concept implementation in here, is not only large in size but also expensive, thus it is upto the vendor or manufacturer to choose an economical board/chip for the STPM design. Therefore, a trusted reconfiguration is the indispensable prerequisite for the realization of reconfigurable trust anchors that allow for sustainable use.

PUBLICATIONS AND THESIS SUPERVISIONS

Following is the list of authored and co-authored publications, which are produced during this thesis work.

[1] S. Malipatlolla, T. Feller, and S.A. Huss. An Adaptive System Architecture for Mitigating Asymmetric Cryptography Weaknesses on TPMs. In Proceedings of IEEE/ACM International Conference on Adaptive Hardware and Systems (AHS 2012), June 2012.

[2] S. Malipatlolla, T. Feller, A. Shoufan, T. Arul, and S.A. Huss. A Novel Architecture for a Secure Update of Cryptographic Engines on a Trusted Platform Module. In Proceedings of IEEE International Symposium on Field Programmable Technology (FPT 2011), December 2011.

[3] S. Malipatlolla and S.A. Huss. A Novel Method for Secure Intellectual Property Deployment in Embedded Systems. In IEEE International Conference on Southern Programmable Logic (SPL 2011), April 2011.

[4] T. Feller, S. Malipatlolla, D. Meister, and S.A. Huss. TinyTPM: A Lightweight Module aimed to IP Protection and Trusted Embedded Platforms. In Proceedings of IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2011), June 2011.

[5] T. Feller, S. Malipatlolla, M. Kasper, and S.A. Huss dcTPM: A Generic Architecture for Dynamic Context Management. IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig 2011), November 2011.

[6] A. Seffrin, S. Malipatlolla, and S.A. Huss. A Novel Design Flow for Tamper-Resistant Self-Healing Properties of FPGA Devices without Configuration Readback Capability. In IEEE International Conference on Field-Programmable Technology (FPT 2010), December 2010.

[7] M. Stoettinger, S. Malipatlolla, and Q. Tian. Survey of Methods to Improve Side-Channel Resistance on Partial Reconfigurable Platforms. In Design Methodologies for Secure Embedded Systems, November 2010.

[8] S. Malipatlolla and I. Stierand. Evaluating the Impact of Integrating a Security Module on the Real-Time Properties of a System. In Proceedings of International Embedded Systems Symposium (IFIP/Springer), June 2013 (to be published).

Following is the list of supervised Theses and Internships during this thesis work.

[1] Minakshi Karwa. Random Number Generators on FPGAs Student Internship (Co-supervised with T.Feller)

[2] Florian Benz. Reverse Engineering the FPGA Bitstream Format Master Thesis (Co-supervised with A.Seffrin)

[3] Mehmet Ariman. Automatic VHDL Testbench Generation Bachelor Thesis (Co-supervised with T.Feller)

[4] Suraj Das. IP-Core Library Framework for Cryptographic Modules Student Internship (Co-supervised with T.Feller)

[5] David Meister, Aziz Demirezen. Tiny TPM - reducing a TPM to its lightweight components Research Work (Co-supervised with T.Feller)

BIBLIOGRAPHY

[1] Secure Execution Mode. Available at http://arstechnica.com/information-technology/2012/06/amd-to-add-arm-processors-to-boost-chip-security/.

[2] Trusted Execution Technology. Available at http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-overview.html.

[3] Microsoft's Next-Generation Secure Computing Base, 2002. Available at http://www.microsoft.com/resources/ngscb/default.mspx.

[4] SmartFusion2 SoC FPGAs: Security - Reliability - Low Power, 2012. Available at http://www.actel.com/documents/SmartFusion2_PIB.pdf.

[5] ML505/ML506/ML507 Evaluation Platform User Guide, 2009. Available at www.xilinx.com/support/documentation/boards_and.../ug347.pdf.

[6] Trusted Computing. Available at http://www.trustedcomputinggroup.org/trusted_computing.

[7] Trusted Computing: Promise and Risk, 2003. Available at https://www.eff.org/wp/trusted-computing-promise-and-risk.

[8] Trusted Platform Module. Available at http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[9] MultiBoot with Virtex-5 FPGAs and Platform Flash XL, 2008. Available at www.xilinx.com/support/documentation/application.../xapp1100.pdf.

[10] Online Banking per HBCI unsicher, 2000. Available at http://www.zdnet.de/news/2051706/online-banking-per-hbci-unsicher.htm.

[11] A. Shoufan. Layered Abstraction Model for Trusted Computing, 2011. The figure resulted during a research discussion of STCG.

[12] AMD. AMD Secure Virtual Machine Architecture Reference Manual, 2005. Available at http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf.

[13] R. Anderson and M. Kühn. Tamper resistance: a cautionary note. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–1, 1996.

[14] R. Anderson and M. Kühn. Low cost attacks on tamper resistant devices. pages 125–136. Springer-Verlag, 1997.

[15] W.B. Andrew, G. Carl, R.K. Charath, J.F. Hoff, R. Modo, H. Nguyen, W. Smith, D. Rhein, J. Schulingkamp, CW Spivak, et al. A field programmable system chip which combines fpga and asic circuitry. In *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999*, pages 183–186. IEEE, 1999.

[16] Anoop MS. Elliptic Curve Cryptography: An Implementation Guide, 2011. Available at http://www.tataelxsi.com/whitepapers/ECC_Tut_v1_0.pdf.

[17] William A. Arbaugh. The TCPA: What is wrong, What is right, and what to do about, 2002. Available at http://www.cs.umd.edu/~waa/TCPA/TCPA-goodnbad.html.

[18] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal 2004-11-17. Technical report, Aalborg University, Denmark, November 2004.

[19] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *Proceedings of Advances in Cryptology (CRYPTO'06)*, Lecture Notes in Computer Science, pages 602–619. Springer-Verlag, 2006.

[20] L. Bolotnyy and G. Robins. Physically unclonable function-based security and privacy in rfid systems. In *Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on*, march 2007.

[21] L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for SRAM FPGA bitstreams. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004*, pages 146–153, Santa Fe, NM, USA. doi: 10.1109/IPDPS.2004.1303128.

[22] H. Brandl. Trusted Computing: The TCG Trusted Platform Module Specification. http://www.infineon.com/dgdl/Basic+Knowledge+EC2004.pdf, 2004.

[23] D. Bruschi, L. Cavallaro, A. Lanzi, and M. Monga. Replay attack in TCG specification and solution. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 11–137. IEEE Computer Society, 2005.

[24] G. Canivet, R. Leveugle, J. Clediere, F. Valette, and M. Renaudin. Characterization of Effective Laser Spots during Attacks in the Configuration of a Virtex-II FPGA. In *Proceedings of the VLSI Test Symposium, 2009. VTS'09. 27th IEEE*, pages 327–332. IEEE, 2009.

[25] L. Chen and M. Ryan. Offline dictionary attack on TCG TPM weak authorisation data, and solution. In *Proceedings of the International Conference on Future of Trust in Computing*, 2008.

[26] L. Chen and M. Ryan. Attack, solution and verification for shared authorisation data in TCG TPM. *Proceedings of the 6th International Workshop on Formal Aspects in Security and Trust*, pages 201–216, 2009.

[27] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[28] Wave Systems Corp. FDIC report finds trusted platform module (tpm) technology provides one alternative for mitigating ID theft in financial services industry, 2005.

[29] Xilinx Corporation. Virtex-6 FPGA Configuration User Guide, November 2011. Available at www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[30] Deichmann, F. Architektur eines Multi Context TPM. Diploma Thesis at CASED Institute, Darmstadt, 2010.

[31] J.P. Deschamps, G.D. Sutter, and E. Cantó. Finite-field arithmetic. *Guide to FPGA Implementation of Arithmetic Functions*, pages 337–355, 2012.

[32] Henning Dierks, Alexander Metzner, and Ingo Stierand. Efficient Model-Checking for Real-Time Task Networks. In *International Conference on Embedded Software and Systems (ICESS)*, 2009.

[33] J. Docherty and A. Koelmans. Hardware implementation of sha-1 and sha-2 hash functions. 2011.

[34] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[35] eASIC Corporation. eASIC nextreme 90 nm NEW ASICs, February 2011.

[36] T. Eisenbarth, T. Guneysu, C. Paar, A.R. Sadeghi, M. Wolf, and R. Tessier. Establishing Chain of Trust in Reconfigurable Hardware. In *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '07, pages 289–290. IEEE Computer Society, 2007.

[37] J. Ekberg and M. Kylänpää. Mobile Trusted Module (MTM) - an Introduction. Technical Report NRC-TR-2007-015, Nokia Research Center, Helsinki, November 2007. Available at http://research.nokia.com/files/NRCTR2007015.pdf.

[38] Eric Conrad. Advanced Encryption Standard, 2011. Available at http://www.giac.org/cissp-papers/42.pdf.

[39] F. Deichmann. Proposed STPM Architecture, 2011. The figure is a modification of conventional TPM architecture.

[40] T. Feller, S. Malipatlolla, D. Meister, and S.A. Huss. TinyTPM: A Lightweight Module aimed to IP Protection and Trusted Embedded Platforms. In *Proceedings of IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2011)*, June 2011.

[41] N. Ferguson. AES-CBC + elephant diffuser: A disk encryption algorithm for windows vista, 2006. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.1617.

[42] T. Finke, M. Gebhardt, and W. Schindler. A New Side-Channel Attack on RSA Prime Generation. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 141–155. Springer-Verlag, 2009.

[43] PUB FIPS. 180-1, Secure Hash Standard, SHA-1. *National Institute of Standards and Technology (NIST)*, 1995.

[44] FxJ. Trusted Platform Module on Asus motherboard P5Q PREMIUM, 2009. Available at http://en.wikipedia.org/wiki/Trusted_Platform_Module.

[45] E. Gallery and C. J. Mitchell. Foundations of security analysis and design iv. chapter Trusted mobile platforms, pages 282–323. Springer-Verlag, Berlin, Heidelberg, 2007.

[46] B. Glas, A. Klimm, O. Sander, K. Muller-Glaser, and J. Becker. A System Architecture for Reconfigurable Trusted Platforms. In *Proceedings of Conference on Design, Automation and Test in Europe*, DATE '08, pages 541–544. ACM, 2008.

[47] J. Guajardo, S.S. Kumar, G. Schrijen, and P. Tuyls. Physical unclonable functions and Public-Key crypto for FPGA IP protection. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 189–195, 2007. doi: 10.1109/FPL.2007.4380646.

[48] T. Guneysu, B. Moller, and C. Paar. Dynamic intellectual property protection for reconfigurable devices. In *Proceedings of the International Conference on Field Programmable Technology*, pages 169–176, 2007. doi: 10.1109/FPT.2007.4439246.

[49] S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing. *Proceedings of the 26th annual international conference on Advances in Cryptology*, pages 41–59, 2006.

[50] A. Hamann, M. Jersak, K. Richter, and R. Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. *Real-Time Systems*, pages 101–137, July 2006.

[51] Intel Corporation. Intel Low Pin Count (LPC) Interface Specification, Revision 1.1, 2002. Available at www.intel.com/design/chipsets/industry/25128901.pdf.

[52] Intrinsic-ID B.V. Quiddikey in Hardware, . Available at http://www.intrinsic-id.com/quiddikey_in_hardware.htm.

[53] Intrinsic-ID B.V. How to choose the right security protocol?, . Available at http://securityblog.astida.com/2009/11/02/how-to-choose-the-right-security-protocol/.

[54] F.K. Jondral. Software-defined radio: basics and evolution to cognitive radio. *EURASIP Journal on Wireless Communications and Networking*, 2005(3):275–283, 2005.

[55] T. Kean. Secure configuration of field programmable gate arrays. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 142–151. Springer-Verlag, 2001.

[56] S. Kinney. *Trusted Platform Module Basics*. Elsevier, 2006. ISBN 0-7506-7960-3.

[57] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):pp. 203–209, 1987. ISSN 00255718.

[58] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.

[59] U. Kühn, K. Kursawe, S. Lucks, A.R. Sadeghi, and C. Stüble. Secure Data Management in Trusted Computing. In *Proceedings of Cryptographic Hardware and Embedded Systems*, Lecture Notes in Computer Science, pages 324–338. Springer Berlin / Heidelberg, 2005.

[60] K. Kursawe and D. Schellekens. Flexible µTPMs through disembedding. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 116–124. ACM, 2009.

[61] J. Lach, H.M. William, and P. Miodrag. Signature hiding techniques for fpga intellectual property protection. In *Proceedings of the IEEE/ACM International Conference on Computer Aided design*, 1998.

[62] Man Lin, Li Xu, L.T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. Static security optimization for real-time systems. *IEEE Transactions on Industrial Informatics*, 5(1):22–37, February 2009. ISSN 1551-3203. doi: 10.1109/TII.2009.2014055.

[63] S. Malipatlolla and S.A. Huss. A Novel Method for Secure Intellectual Property Deployment in Embedded Systems. In *IEEE International Conference on Southern Programmable Logic (SPL 2011)*, April 2011.

[64] S. Malipatlolla, T. Feller, A. Shoufan, T. Arul, and S.A. Huss. A Novel Architecture for a Secure Update of Cryptographic Engines on a Trusted Platform Module. In *Proceedings of IEEE International Symposium on Field Programmable Technology (FPT'11)*, December 2011.

[65] S. Malipatlolla, T. Feller, and S.A. Huss. An Adaptive System Architecture for Mitigating Asymmetric Cryptography Weaknesses on TPMs. In *Proceedings of IEEE/ACM International Conference on Adaptive Hardware and Systems (AHS'12)*, June 2012.

[66] S. Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Proceedings of the 5th international conference on Information security and cryptology*, pages 343–358, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-00716-4.

[67] Meister, D. Readback and reconfiguration wit the ICAP interface. Research study at CASED Institute, Darmstadt, 2011.

[68] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC, 1996.

[69] National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, 2005.

[70] National Institute of Standards and Technology. Digital Signature Standard (DSS). Technical report, National Institute of Standards and Technology, June 2009.

[71] National Institute of Standards and Technology. *Recommendation for the Transitioning of Cryptographic Algorithms and Key Lengths*, 2010.

[72] National Security Agency. The case for elliptic curve cryptography, 2009. Available at http://www.nsa.gov/business/programs/elliptic_curve.shtml.

[73] J. Note and E. Rannaud. From the bitstream to the netlist. In *Proceedings of the 16th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '08, pages 264–264. ACM, 2008.

[74] Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria, 1985. Available at http://csrc.nist.gov/publications/history/dod85.pdf.

[75] T. Popp. An Introduction to Implementation Attacks and Countermeasures. In *Proceedings of IEEE/ACM International Conference on Formal Methods and Models for Co-Design*, MEMOCODE'09, pages 108 –115. IEEE Press, July 2009.

[76] Preneel, B. Upgrading Cryptographic Algorithms for Network Security, 2009. Available at http://homes.esat.kuleuven.be/~preneel/preneel_securecom09.pdf.

[77] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser. An MDE Approach for Implementing Partial Dynamic Reconfiguration in FPGAs. In *16th International Conference on IP Based System-on-chip*, 2007.

[78] Renesas. Preliminary Application Note Renesas AN0303011.

[79] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[80] A.R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside?: A note on TPM specification compliance. In *Proceedings of the first ACM workshop on Scalable trusted computing*, STC '06, pages 47–56. ACM, 2006.

[81] D. Schellekens, P. Tuyls, and Preneel B. Embedded Trusted Computing with Authenticated Non-Volatile Memory. In *Proceedings of 1st international conference on Trusted Computing and Trust in Information Technologies:*

Trusted Computing - Challenges and Applications, Trust '08, pages 60–74. Springer-Verlag, 2008.

[82] E. Simpson and P. Schaumont. Offline Hardware/Software authentication for reconfigurable platforms. In *Proceedings of the Cryptographic Hardware and Embedded Systems CHES 2006*, pages 311–323, 2006.

[83] S. W. Smith. *Trusted Computing Platforms: Design and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004. ISBN 0387239162.

[84] A. Spalka, A. B. Cremers, and H. Langweg. Protecting the creation of digital signatures with trusted computing platform technology against attacks by trojan horse programs. In *Proceedings of the 16th international conference on Information security: Trusted information: the new decade challenge*, pages 403–419, 2001.

[85] M. Stoettinger, S. Malipatlolla, and Q. Tian. Survey of Methods to Improve Side-Channel Resistance on Partial Reconfigurable Platforms. In *Design Methodologies for Secure Embedded Systems*, November 2010.

[86] M. Strasser, H. Stamer, and J. Molina. Software-based TPM Emulator. *ETH Zurich. Dostupné na http://tpm-emulator. berlios. de/index. html (prosinec 2009)*.

[87] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07. ACM, 2007.

[88] Trusted Computing Group, Incorporated. Features under consideration for the next generation of tpm, 2009. Available at http://www.trustedcomputinggroup.org/resources/summary_of_features_under_consideration_for_the_next_generation_of_tpm.

[89] Trusted Computing Group, Incorporated. TCG protection profile pc client specific trusted platform module tpm family 1.2; level 2, 2008. Available at http://www.commoncriteriaportal.org/files/ppfiles/pp0030b.pdf.

[90] Trusted Computing Group, Incorporated. TCG software stack TSS specification version 1.2, 2006. Available at http://www.trustedcomputinggroup.org/files/resource_files/647A8E37-1D09-3519-AD860BD090352A8B/TSS_Version_1.2_Level_1_FINAL.pdf.

[91] Trusted Computing Group, Incorporated. TCG specification architecture overview, 2007. Available at http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14.

[92] Trusted Computing Group, Incorporated. *TCG Mobile Trusted Module Specification Version 1.0*, June 2008.

[93] H.J. Tsai. Microcontroller with programmable embedded flash memory, 1999. US Patent 6,009,496.

[94] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel Virtualization Technology. *Computer*, 38(5):48 – 56, may 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.163.

[95] University of Bourgogne. The RSA Algorithm, 2011. Available at http://math.u-bourgogne.fr/MASTERMIGS/etudiants/MIGS2/Fichiers/Anglais/ExposeCeline.pdf.

[96] U.S. Department of Transportation. Preliminary Report:The Incidence Rate of Odometer Fraud, 2002. Available at http://www.nhtsa.gov/cars/rules/regrev/evaluate/pdf/809441.pdf.

[97] John A. W. uClinux for Microblaze, 2009. Available at www.petalogix.com.

[98] X. Wang, Y.L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Proceedings of the 25th annual international conference on Advances in Cryptology*, CRYPTO'05, pages 17–36. Springer-Verlag, 2005.

[99] Marko Wolf and Timo Gendrullis. Design, implementation, and evaluation of a vehicular hardware security module. In Howon Kim, editor, *Information Security and Cryptology - ICISC 2011*, volume 7259 of *Lecture Notes in Computer Science*, pages 302–318. Springer Berlin Heidelberg, 2012.

[100] K.C. Wu and Y.W. Tsai. Structured ASIC, evolution or revolution? In *Proceedings of the 2004 International Symposium on Physical Design*, ISPD '04, pages 103–106. ACM, 2004.

[101] xen.org. xen hypervisor, 2011. Available at http://www.xen.org/products/xenhyp.html.

[102] Xilinx Corporation. System ACE CompactFlash Solution, 2008. Available at http://www.xilinx.com/support/documentation/data_sheets/ds080.pdf.

[103] Xilinx Corporation. MicroBlaze Processor Reference Guide, 2012. Available at http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf.

[104] Xilinx Corporation. LogiCORE IP MicroBlaze MicroController System v1.1, 2012. Available at http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf.

[105] Xilinx Corporation. Partial Reconfiguration User Guide, September 2009.

[106] Xilinx Corporation. Spartan-3AN FPGA Family Data Sheet, 2011. Available at http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf.

[107] Xilinx Corporation. Virtex-5 FPGA Configuration User Guide, 2011. Available at http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

[108] Xilinx Corporation. Xilinx DS100 Virtex-5 Family Overview, February 2009.

[109] K. Yip and T. Ng. Partial-encryption technique for intellectual property protection of FPGA-based products. *IEEE Transactions on Consumer Electronics*, 46(1):183–190, February 2000. ISSN 00983063. doi: 10.1109/30.826397.

[110] P.S. Zuchowski, C.B. Reynolds, R.J. Grupp, S.G. Davis, B. Cremen, and B. Troxel. A hybrid asic and fpga architecture. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 187–194. IEEE, 2002.

ERKLÄRUNG ZUR DISSERTATION

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Darmstadt, den 01 Mai 2013 Sunil Dath K. Malipatlolla

Curriculum Vitae

Sunil Dath Kumar Malipatlolla
Donnerschweer Str. 218
26123 Oldenburg, Germany
Email: sunil.malipatlolla@offis.de
Phone: +49 (0)176 22793980

Summary

Work experience	6 **years** in FPGA-based HW/SW design and Hardware Security
Areas of Interest	Trusted Computing, Embedded System Security, and FPGAs

Education

Jul 09 – Jun 12	**Doctor of Philosophy (Ph.D)** Center for Advanced Security Research Darmstadt (CASED) and Technische Universität Darmstadt, Germany Title: Sustainable Trusted Computing: A Novel Approach for a Flexible and Secure Update of Cryptographic Engines on a Trusted Platform Module Grade: **Passed**
Apr 06 – Sept 08	**Master of Science (M.Sc.)** Friedrich Alexander Universität, Erlangen-Nürnberg, Germany Subject: Computational Engineering Specialization: Microelectronics Average Grade: **1.8** (Scale: 1.0 to 5.0, Best: 1.0)
Nov 00 – May 04	**Bachelor of Technology (First class with Distinction)** Jawaharlal Nehru Technological University, Hyderabad, India Specialization: Electronics and Communication Engineering Average Grade: **80.04%**

Work Experience

Aug 12 – to date	**OFFIS-Institute for Information Technology**, Germany Position: Scientific Researcher - Security in real-time embedded systems
Jul 09 – Jun 2012	**Center for Advanced Security Research Darmstadt**, **and Technische Universität Darmstadt,** Germany Position: Doctoral Researcher - Designed and implemented an architecture for providing sustainable features for a trusted platform module - Supervised Interns and Bachelor Theses
Apr 07– Mar 09	**Friedrich Alexander Universität,** Erlangen-Nürnberg, Germany as Graduate Research Assistant at chair for HSCD

- Developed a communication interface between an external memory placed on a FPGA board and the software running on the processor
- Master Thesis: Developed an HW/SW Co-designed optimized implementation for the Blocked Matrix Multiplication application using Xilinx EDK Tool

Jun 04 – Feb 06 **TRR College of Engineering,** Hyderabad, India
Lecturer at Electronics and Communications Engineering Group

- Courses taught at undergraduate level:
Digital Design, Microprocessors and Microcontrollers

Academic Projects

Jul 09 – Jun 12 **Doctoral Thesis**, Center for Advanced Security Research Darmstadt, Germany
- Title: Sustainable Trusted Computing: A Novel Architecture for a Flexible and Secure Update of Cryptographic Engines on a Trusted Platform Module

Apr 08 – Sep 08 **Master Thesis**, Friedrich Alexander Universität, Erlangen-Nürnberg, Germany
- Title: Automatic Interface Generation for the Integration of Hardware Accelerators

Additional Qualifications

Language skills	Native - Marathi Fluent- English, Hindi, Telugu Intermediate - German
Computer skills	VHDL, C/C++, Xilinx ISE Suite, Linux, Latex, Windows, Matlab FPGA architecture design and implementation, Embedded system design, Hardware Security
Other skills	Self-motivated, Project planning and tracking Effective team player, Excellent communication capabilities

Awards and Achievements

Scholarship	Recipient of **Siemens Scholarship** for Master Studies in Germany
Academic rankings	Ranked in top 3 out of 60 students in Bachelor studies

Personal Data

First Name	Sunil Dath Kumar
Last Name	Malipatlolla
Date of birth	11.06.1982
Nationality	Indian
Gender	Male
Hobbies & Interests	Cricket, Reading, Traveling and Music.

References

Will be made available upon request