Secure Reprogramming of a Network Connected Device

Securing programmable logic controllers

MUSSIE TESFAYE

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Secure Reprogramming of a Network Connected Device

Securing programmable logic controllers

Mussie Tesfaye

mtesfaye@kth.se

2012.10.25

Academic adviser and examiner: Prof. Gerald Q. Maguire Jr.

Communication Systems
School of Information and Communications Technology
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

This is a master's thesis project entitled "Secure reprogramming of network connected devices". The thesis begins by providing some background information to enable the reader to understand the current vulnerabilities of network-connected devices, specifically with regard to cyber security and data integrity. Today supervisory control and data acquisition systems utilizing network connected programmable logic controllers are widely used in many industries and critical infrastructures. These network-attached devices have been under increasing attack for some time by malicious attackers (including in some cases possibly government supported efforts).

This thesis evaluates currently available solutions to mitigate these attacks. Based upon this evaluation a new solution based on the Trusted Computing Group (TCG's) Trusted Platform Modules (TPM) specification is proposed. This solution utilizes a lightweight version of TPM and TCG's Reliable Computing Machine (RCM) to achieve the desired security. The security of the proposed solution is evaluated both theoretically and using a prototype. This evaluation shows that the proposed solution helps to a great extent to mitigate the previously observed vulnerabilities when reprogramming network connected devices.

The main result of this thesis project is a secure way of reprogramming these network attached devices so that only *a valid* user can successfully reprogram the device and *no one else can reprogram the device* (either to return it to an earlier state, perhaps with a known attack vector, or even worse prevent a valid user from programming the device).

Keyword: SCADA, PLC, SCADA security, SCADA networks, PLC security, Trusted computing, TCM, TPM, Embedded security, Digital security.

Sammanfattning

Avhandlingen börjar med att ge lite bakgrundsinformation för att läsaren att förstå de nuvarande sårbarheten i nätverksanslutna enheter, särskilt när det gäller IT-säkerhet och dataintegritet. Idag övervakande kontroll och datainsamlingssystem använder nätverksanslutna programmerbara styrsystem används allmänt i många branscher och kritisk infrastruktur. Dessa nätverk anslutna enheter har under ökande attacker under en tid av illvilliga angripare (inklusive i vissa fall eventuellt regeringen stöds insatser).

Denna avhandling utvärderar för närvarande tillgängliga lösningar för att minska dessa attacker. Baserat på denna utvärdering en ny lösning baserad på Trusted Computing Group (TCG) Trusted Platform Modules (TPM) specifikation föreslås. Denna lösning använder en lätt version av TPM och TCG:s pålitliga dator (RCM) för att uppnå önskad säkerhet. Säkerheten i den föreslagna lösningen utvärderas både teoretiskt och med hjälp av en prototyp. Utvärderingen visar att den föreslagna lösningen bidrar i stor utsträckning för att minska de tidigare observerade sårbarheter när omprogrammering nätverksanslutna enheter.

Huvudresultatet av denna avhandling projektet är ett säkert sätt omprogrammering dessa nätverksanslutna enheter så att endast ett giltigt användarnamn framgångsrikt kan omprogrammera enheten och ingen annan kan programmera enheten (antingen att återställa den till ett tidigare tillstånd, kanske med en känd attack vector, eller ännu värre förhindra en giltig användare från programmering av enheten).

Nyckelord:*SCADA, PLC, SCADA säkerhet, SCADA-nätverk, PLC säkerhet, pålitlig datoranvändning, TCM, TPM, inbäddad säkerhetlösningar, datasäkerhet*

Acknowledgement

The events leading to this point has been full of unexpected surprise and the time had never been easy. Even though I would like to thank all those, especially professor Gerald Q. “Chip” Maguire Jr. and my family, who has helped me in one way or the other. I know my simple words cannot express the sincere and humble gratitude I feel.

I would like to take this opportunity to express my heart felt gratitude and appreciation to professor Gerald Q. “Chip” Maguire Jr., my examiner. His advice and feedback was not only been invaluable, but also has been a personal motivation and corner stone of my work. He has helped me express my thoughts more clearly and inspired me to aim higher. He never hesitated to read and comment on my work nor did he hold back from giving me important advice or answering my questions. Every meeting I had with him inspired me to work more and harder. Thank you for everything and may the Almighty God bless you for your help.

I would also like to extend my gratitude to my family and friends who have always supported and comforted me. I would like to thank you all for standing by me when I needed help and for believing in me when I had lost all hope. I am forever in debt for all the things you did.

Sincerely,

Mussie Tesfaye

Table of Contents

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACL	Access control list
AGA	American Gas Association
AIK	Attestation Identity Key
CA	Certification Authorities
DCS	Distributed control system
DIR	Data Integrity Registers
DMA	Direct Memory Access
DoD	(United States of America) Department of Defense
EEPROM	Electrically Erasable Programmable ROM
EK	Endorsement Key
HMI	Human-machine interfaces
I/O	Input or Output
IP	Internet protocol
ISA	International Society of Automation
ISO	International Organization for Standardization
OS	Operating System
OSI	Open Systems Interconnection
PCR	Platform Configuration Register
PKI	Public Key Infrastructure
PLC	Programmable logic controller
Prk	Private Key
Puk	Public Key
RAM	Random Access Memory
RCM	Reliable Computing Machine
RNG	Random Number Generator
ROM	Read Only Memory
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman encryption
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SBL	Secured Boot Loader
SCADA	Supervisory Control And Data Acquisition
SHA	Secured Hashing Algorithm
SK	Secret Key equivalent to EK
SP	Secured Protocol
TCG	Trusted Computing Group

TCM	Trusted Computing Module
TCP	Transport Control Protocol
TCPA	Trusted computing platform alliance
TPM	Trusted Platform Modules
UDP	User datagram protocol
VFD	Variable frequency drive
VHDL	VHSIC Hardware description language

1 Introduction

A supervisory control and data acquisition system (SCADA) is a special computer network connecting one or more devices that are used to control and monitor industrial, infrastructure, and facility based processes[1]. SCADA systems are used for controlling and monitoring industrial facilities, such as manufacturing plants, nuclear or coal burning power plants, wind farms, water treatment, and refineries. SCADA systems are also used to monitor and control infrastructures, such as electrical power distribution grids, natural gas and oil pipeline systems, and water supply systems.

A SCADA system includes a network of devices with sensors and actuators occupying the bottom of a hierarchy. These sensors and actuators are connected to programmable logic controllers (PLCs). A PLC is used to control the actuator(s) based on the information gathered by sensors under the control of logic running in the CPU of the PLC. These PLCs are connected either to other PLCs or to a command and control station. The command and control stations usually provide a human-machine interface (HMI). Using this HMI, users can program and control the PLCs. A typical industrial SCADA system is shown in Figure 1.

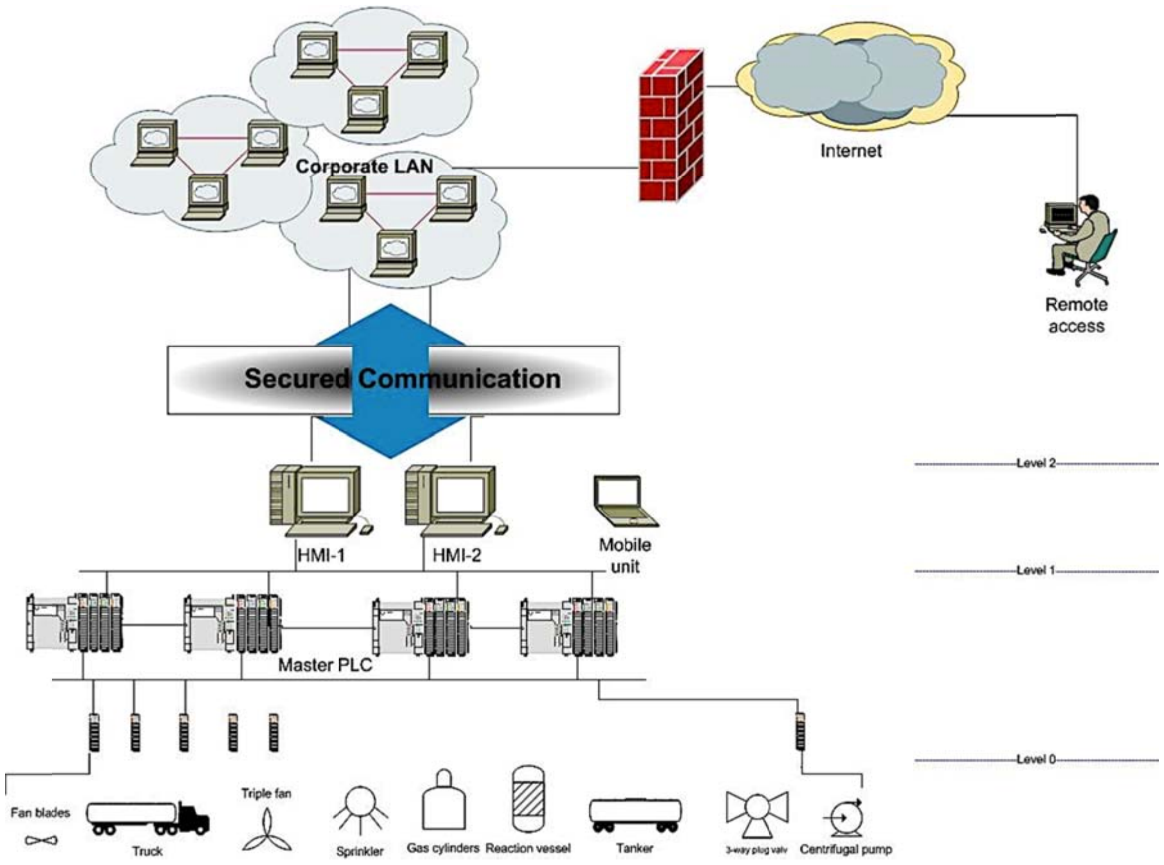


Figure 1: Typical SCADA network

The international standard for developing an automated interface between enterprises and control systems has been defined by the International Society of Automation (ISA) in ISA-97[2]. This standard provides the overall architecture for production modeling. ISA-95 defines a five-level hierarchical model and lists components and services to be categorized into each level. These five levels form a triangle with level 0 at the broad base and level 4 at the tip of the triangle. The first three of these levels were shown in Figure 1. The five levels are shown in Table 1. A security breach at level 2 could bring down the whole production process, halt its operation, and/or cause severe danger to the plant and/or human beings.

Table 1: Five level ISA-97 model

Level 0	*Physical level* containing sensors, actuators and process equipment, such as motors and valves.
Level 1	*Programmable logic controllers* working on sensor output, commands for actuators, and communicating with higher levels and other level-1 equipment.
Level 2	*Supervisory control level (SCADA) and Human-Machine Interface (HMI)* are level 2 equipment and systems that communicate with, control, and monitoring lower level equipment's performance.
Level 3	*Manufacturing operations and control* works with dispatching production, detailed production scheduling, production material selection, modifying production schedule, and locally optimizing resources for individual production processes. This level is also concerned with process management, production planning and tracking, and performance analysis.
Level 4	*Business planning and logistics* level is concerned with plant production scheduling, operational management, and collecting and maintaining overall energy use, raw materials, spare parts, organizing manpower, maintaining and servicing machines, and optimizing the overall plant's operations.

1.1 Problem context

Digital security, virus attacks, and computer hacking have moved from being an interesting theme of a science fiction movie to reality, resulting in loss of personal information and denial of service for major services. Today attacks are being waged as a form of cyber warfare with attacks designed to damage industrial and infrastructure targets.

SCADA systems used to control very critical systems such as power plants, sewerage treatment systems, water treatment plants, and almost all industries have been found to be quite vulnerable to many forms of security breaches. These security breaches occur either because the SCADA networks are connected to a larger corporate network, personnel carry in the attack code (either consciously or unconsciously), or due to vulnerabilities in the operating system on which the HMI applications are running. With the increasing political and social instability in the world, SCADA systems are prime targets for attacks; hence there is an increasing need to secure SCADA systems.

Owing to vulnerabilities in the design, deployment, and use of SCADA systems the number of security breaches to SCADA systems is increasing daily. Most of these incidents do not get into the news for security reasons. Some attacks are not even detected until it is too late[3].

Gathering information about security breaches within such systems is difficult because most security breaches are kept secret. There are those who wish to keep these breaches secret for security reasons, while others wish to keep these breaches secret because they are concerned that the public, their customers, their shareholders, etc. might find the management or workers in the facility to be careless, negligent, or malicious. However, there have been some publically reported incidents, such as the Australian sewage release incident where a disgruntled employee accessed the sewage SCADA system and released a large amount of sewage into a public area[1,4,5]. In January 2003, an incident occurred where a Slammer worm bypassed a network firewall and disabled the safety monitoring system of the Davis-Basse nuclear power plant in Oak Harbor, Ohio, USA for six hours, resulting in the plant being shutdown to avoid an accident[6]. In April 2007, the United States Nuclear Regulatory Commission reported it had to shutdown a nuclear reactor (Unit 3) at the Browns Ferry nuclear plant near Athens, Alabama, USA because the safety variable frequency drive (VFD) stopped responding due to excessive traffic on the SCADA network[7].

Stuxnet is a computer worm that was designed to affect industrial control systems. The worm managed to attack several industrial control systems in Iran, Indonesia, and India until it was exposed in 2010. In September 2010, Iran admitted that the Stuxnet worm had infected the Bushehr nuclear reactor facility causing a major setback in their plans for uranium enrichment[8,9]. This worm infected around 30,000 computers in Iran and managed to reprogram PLCs connected to centrifuges in the nuclear plant causing them to spin out of control[10].

The Stuxnet worm exploited publically known and unknown vulnerabilities in Microsoft Windows to copy itself from one PC to another through the network and infected removable media. It attaches itself to Step 7 (the Siemens PLC configuration and management tool) and whenever the PLC programmer application is launched the worm is executed. It formed a peer-to-peer (P2P) network to upgrade itself and exchange SCADA system information with the attacker. It managed to bypass security products and utilized a rootkit to gain privileged access to the system and its resources. The worm replicated itself through the network searching for HMI's connected to specific Siemens PLCs controlling motors running at a specific frequency. When a target was acquired the worm modified the Step 7 program sent from the HMI to the PLC to cause the motors to spin at varying rates *without* causing any alarms to occur. Had it not been detected, the worm was designed to delete itself at a specific time so that no one would have known it had existed. The Stuxnet worm was seen as a wakeup call to the SCADA security community and it was my personal motivation for undertaking this thesis project. Further details of the Stuxnet work can be found in [3, 10-12].

Stuxnet is only one example of a worm targeting SCADA systems; there are many other unreported intrusions and attacks. Despite the knowledge that attacks are taking place against SCADA systems, today SCADA networks are still vulnerable. The vulnerability in SCADA systems is alarming and devising a single solution that would address all the vulnerabilities might be impossible. However, combining different security solutions at different levels of the production model[2] could help overcome at least some of these problems.

The above are only a few of the incidents that have made the public concerning about the security of SCADA systems and the effects of a successful attack. Even more disconcerting is that most incidents are **not** reported. How to secure these vulnerable and mission critical systems remains an unanswered question.

1.2 Problem Statement

This master's thesis will investigate state of the art solutions for securely reprogramming network-connected devices. The thesis will propose a solution and evaluate it theoretically. In addition, the thesis will describe the implementation and evaluation of a prototype of the proposed solution.

As mentioned above, SCADA systems are frequently used to control very critical and essential systems. Unfortunately, these systems are currently vulnerable to external attacks and exploits – as most such systems were initially designed with the expectation that they would be completely isolated, i.e., that there would be no connection to external networks.

One of the most devastating attacks on SCADA system involves maliciously reprogramming the PLCs, as in this way the attacker can perform an attack where the actual effects of the attack do **not** occur until later – thus making it hard to correlate the attack with the effects of the attack. Additionally, the operator of the SCADA system may need to reprogram specific controllers to correct flaws or add new functionality. Finding a secure way of reprogramming these devices not only means that *a valid* user successfully reprogram the device, but also that *no one else can reprogram the device* (either to return it to an earlier state, perhaps with a known attack vector, or even worse prevent a valid user from programming the device). To fully address this problem we must devise solutions that address the following requirements:

- The devices shall only be programed based upon commands from a valid user. This means that commands sent from a valid user's terminal by malicious software or compromised commands that have penetrated the network's boundary should be **unable** to alter the programming of a device connected to the network.
- Only valid users should be able to program the devices, i.e. neither an imposter assuming our identity on the network *nor* a user without the appropriate privilege can reprogram a device.
- We should be able to program the devices whenever we wish, i.e. no one should be able to block us or keep us from controlling and programming the devices.

These requirements mean that to reprogram the devices we must both have the appropriate privileges and our network needs to be secured from external intrusions. Furthermore, we need to protect the devices on the network from a denial of service attack. To achieve these goals we will study the basic properties of a SCADA network and the state of the art solutions that have been proposed by others. Following this we will propose a solution and later design and evaluate a prototype of our proposed solution.

1.3 Structure of the thesis

The four chapters, chapter 2 to chapter 5, will give a brief background explanation of SCADA components, networks, and protocols. These chapters also address why these systems are so vulnerable and what could be done or what is being done to mitigate these vulnerabilities. This will be followed by an evaluation of existing vulnerabilities and solutions. Then chapter 6 to chapter 8 will present the proposed solution that would help mitigate these vulnerabilities, and also develop and discus the required specifications for the solution. Chapter 9 and chapter 10 will give the analysis and conclusion, respectively. Summary outline of each chapter is given below.

Chapter 2 will give brief introduction of SCADA components. It will start with an emphasis on programmable logic controllers, remote terminal units and intelligent electronic devices then a less detailed introduction of the network components will follow. This chapter will lay the foundation for the paper by answering what network-connected devices mean.

Chapter 3 will give introductory review about digital security. The chapter will dwell on cryptographic methods basic principle followed by brief explanation on certification and authentication. The chapter also gives introduction to the Trusted Computing principles, followed by the brief outline of the trusted computing specifications and couple of implementations done by ARM, and Microsoft. These sections will provide the basic concepts concerning digital security and trusted computing that is needed to understand this paper.

Chapter 4 will briefly introduce the SCADA networks protocols. Here the critical features of some common SCADA protocols will be presented. The chapter is intended to give an insight to the SCADA network and protocols and to learn the strength, weakness, and features of the protocols. The previous two chapters along with this chapter are used to answer the basic question regarding what is network connected devices; what is security in digital world or in network connected devices; and how are these devices connected to the network. With these three chapters the thesis topic "Secured Reprogramming of Network Connected Devices" would be justified.

Chapter 5 will evaluate methods to assure SCADA networks security. To achieve this an evaluation of security risks and causes of security breach will be presented. This will be followed by a summary of best practices that will help to achieve the desired level of security. Finally couple of implemented methods will be presented and compared against the best practice and the vulnerabilities addressed.

Chapter 6 will breakdown the goals and proposes a solution to achieve these goals. Here an overview of the proposed system is presented. This will provide the reader with an overview of the whole system, and individual components proposed here will be presented in the following two chapters.

Chapter 7 will present the hardware implementation proposed in this paper. It will outline the basic components required to implement a lightweight version of the TCM. Each components function, requirement and implementation will be presented. This would give the hardware implementation detail and outline the requirements and propose implementations.

Chapter 8 will present the software components required by the proposed solutions. This will include the software needed to achieve a secured boot, which is a boot loader software running on the main processor; the communication protocol implemented on the proposed hardware; and the software needed to implement a reliable computing machine. A detailed explanation is given on each component in the subsequent subsections.

Chapter 9 will present the analysis and implementation. Here an informal security analysis will be presented where the proposed system is weighed against the best practices and top vulnerabilities proposed in chapter 5. This will be followed by discussion on the implemented prototype and result and measurement section, which will compare the implementation cost and execution cost against a typical PLC. Finally a discussion on the merits and demerits of the proposed system as gathered in the previous sections will be presented.

Chapter 10 will present a conclusion on the overall aspect of "Secure Reprogramming of network connected device" and a discussion on possible future work will be presented. This will be the last chapter followed by reference section and appendix of implementation code.

2 Network Connected Devices

This chapter describes some of the most important elements of SCADA systems, with a focus on the devices that are attached to the network. The chapter begins by describing PLCs, remote terminal units, and intelligent electronic devices. The chapter ends with a description of some of the other devices commonly found in a SCADA network.

2.1 Programmable Logic Controllers (PLC)

A PLC is a special form of a microprocessor-based controller that uses programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting, and arithmetic in order to control machines and processes[13,14]. A PLC is specifically designed to handle a harsh industrial environment, thus it is designed to tolerate and perform despite high/low temperatures, high humidity, and/or strong vibrations. In addition, it is designed to connect, sample, and communicate with digital and analog sensors. A typical PLC can communicate over several different standard networks and protocols. The device is designed so that technicians and programmers can easily program it. These features give a PLC high flexibility, robustness, and reliability, making it the primary means to provide computer control of industrial applications and control systems. PLC applications range from simple motor control to continuous process manufacturing industries and further to monitor & control devices and processes in almost any industry.

PLCs were originally designed to replace relays, timers, and sequencers. Prior to PLCs these other types of devices were used to implement hard-wired control panels. Dick Morley, known as “Father of PLC”, invented the first PLC for General Motors Corporation’s Hydra-Matic division in 1968[15,16]. The original device was similar to a reprogrammable relay, which overcame the problems of fixed, wired relay control systems. The next major leap forward was the introduction of cathode ray tube (CRT) equipped programming devices[14], followed by an expansion in the amount of memory, an increased number of input/output (I/O) interfaces, analog I/O, and serial point to point communication interfaces. Further improvements were introduced with microprocessor based PLCs, local area network interfaces, universal programming devices, and the use of improved redundant architectures. Additional improvements continue to be introduced with regards to better programming languages, improved security, and more flexible and higher data rate communications.

PLCs are applied in almost every industry that exists today due to the PLCs’ unique features and characteristics. According to C. T. Jones, these features are [14]:

- Solid state components,
- Flexible computer based architecture,
- Built to function in industrial environments,
- Programmable stored programs,
- Perform relay equivalent functions and more,
- High modularity,
- Easily built and maintained by plant personnel, and
- Reusable system components.

These features and characteristics have placed PLCs at the heart of almost every industry, thus the vulnerabilities of PLCs can negatively affect nearly every industry!

The core components of any PLC are CPU, memory, I/O interfaces, programming unit, power supply, and communication interfaces. For the purpose of fault tolerance, these components might exit in triples or pairs, so that if the primary component fails there is a backup. In some devices with triple redundancy the components vote to ensure that a malfunctioning component will not affect the correction operation of the system and an alarm can be raised so that maintenance personnel can replace the defective component.

The ***CPU*** is typically a microcontroller with a small *register memory* of a few kilobytes, a *control unit* for timing and generating events, and an interrupt controller to deal with incoming events. The CPU's *arithmetic and logic unit (ALU)* may range from supporting simple integer addition and comparisons operations to floating point operations.

There are several types of ***memory*** used in PLCs to store user data or instructions. Typically a read only memory (ROM) is used to store the operating system and permanently fixed parameters. Random access memory (RAM) is used to store user programs while they are being executed along with their data. This data can include input and output status and values, counters, and timer status and value. Electrically erasable programmable read only memory (EEPROM) is frequently used to store applications and parameters that can be modified. Not surprisingly the RAM and EEPROM will be the main targets of attacks, as they can change the execution of the currently running software or change the execution of software that will later be executed. One of the advantages of making the changes to the contents of the RAM is that when the device is restarted the contents of RAM are generally erased; hence any malicious code that was running in RAM is gone – until the RAM is affected either by loading malicious code or data from EEPROM or via a communication interface. Note that the EEPROM and stable data and program storage might be implemented by flash memories.

I/O interfaces are input or output ports used to input signals to the PLC or drive an external load connected to the PLC. I/O interfaces provide electrical isolation from the external devices, while allowing the PLC to sense and control these external devices. An I/O interface could control a relay (for slow switching), a transistor (for fast direct current (DC) switching), or a triac (for fast alternating current (AC) switching).

The ***power supply unit*** converts industry standard AC (or in some cases DC) power to the appropriate DC voltage level(s) to power the CPU, memory, I/O devices, and communication interfaces.

A ***programming device*** could be special purpose device or standard personal computer (PC) that is used to configure, program, and download programs into the PLC.

A ***communication interface*** is used to communicate with remote PCs or other PLCs for synchronization and communication of values and even programs. The communication interface should support device identification, data acquisition, and connection management[13]. Device identification can range from simply providing information about the type of the device (make and model of the device) to cryptographically signed proofs of unique identities.

2.2 Remote Terminal units (RTU)

RTUs also known as remote telemetry unit are microprocessor based standalone data accusation and control unit used to gather data from remote points and to transfer this data either directly to a central station or via other RTUs that act as relays[17]. RTU can range from small devices having 10-20 digital/analog inputs, through middle range devices having 100 digital and 20-30 analog inputs, to large devices with many more inputs and outputs. An RTU is usually equipped with a radio interface for longer distance communications, CPU for data processing, multiple I/O interfaces for data sensing and control, and a power supply.

The RTU's ***CPU*** is equipped with a microprocessor, RAM, EEPROM/Flash, and ROM. Similar to a PLC; the different memories are used to hold user programs and data. The processor's execution is typically controlled by a scheduler (with potentially a number of different timers) or interrupts.

The RTU's ***I/O interfaces*** generally include *analog* input interfaces that amplify, sample, and quantize analog signals to digital values, while the digital input interfaces count, accumulate, or sequence input signals. Conversely the output interfaces provide electrical isolation, switching, and voltage or current references (for analog outputs) and logical control signals (for digital outputs).

The RTU can have a variety of ***communication interfaces*** providing different link ranges and supporting a variety of communications media, such as [17]:

- Serial communication RS-232/RS-442/RS-485,
- Ethernet,
- Dial up telephone lines/dedicated landlines,
- Microwave/MUX links,
- Satellite links, and
- Radio via trunked/VHF/UHF/900 MHz.

The RTU's ***power supply*** unit usually supplies the RTU with one or more DC voltages either by converting input from an AC supply or by regulating and managing a battery. For reliable operation the RTU may combine both types of power sources. The RTU might utilize the battery either as a backup power source or as the main power source, while the AC mains power is either the primary power source or simply used to recharge the battery.

An RTU is quite useful for remote and distributed systems. Each RTU can be managed and controlled from a convenient central station. RTUs are usually found in application areas, such as offshore oil and gas sites where the sites are located in the middle of a sea and the control station is located hundreds of miles away on the shore. Other application of RTUs include waste water treatment controlling a network of pumps and valves, environmental monitoring, and the like[18].

2.3 Intelligent Electronic Devices (IEDs)

An IED is a core component in overhead (power transmission) line protection and management for substations and power station systems. It is referred as 'intelligent' because it has local intelligence to control and make local decisions. Similar to an RTU or PLC, IDEs are microprocessor-based devices, which can directly communicate with a SCADA system.

IEDs represent a broad range of devices, but all IEDs must fulfill the following functions:

- Provide (line) protection: The most common types of ***protection*** are against different current, voltage, or earth faults, auto-re-closure faults, and different frequency faults.
- Provide fully programmable local and remote ***control,*** including control sequencing, breaker isolation, and generating status and information alarms.
- M***onitoring*** internal and external events, such as: relay temperature and circuit-breaker condition.
- ***Meter*** different parameters by measuring current, voltage, frequency, etc.
- Support both upper level SCADA ***communication*** and direct serial or optical ***communication*** for configuration and uploading of data.

2.4 Other components

This section describes a number of other components that are frequently connected to a SCADA network.

2.4.1 Routers

In the context of a SCADA network a router is a device that connects different networks, such as the SCADA network, LANs, and Internet access links – generally using the Internet protocol (IP). Routers provide inter-network communication and isolate the individual networks by using routing tables or routing polices. There are different types of industrial grade routers ranging from those with cable and optical interfaces to those connecting via a wide area cellular network (such as GSM, UMTS, LTE, etc.) or satellite link.

2.4.2 Firewalls

Firewalls are used to protect networks from outside intrusion by only allowing certain packets to enter the network and denying other packets entrance to the network form beyond the firewall. Packet filtering, network address translation, and user privilege checking are some of the mechanisms implemented by firewalls to provide security against data theft and various types of attacks[19].

2.4.3 Terminals

Terminals are special purpose computers used to configure and program PLCs or RTUs. These devices are equipped with a high-speed processor and RAM to facilitate reprogramming, setting configuration parameters, monitoring network connected devices, etc. These terminals typically are running one of the common operating systems, such as Microsoft's Windows or Linux. Terminals integrated with the system and used to monitor and configure the system provide operators of the plant or other facility with a HMI.

3 Digital Security

The basic idea behind digital security is to protect the system, to protect personal information, and to protect resources in the system from unauthorized access while maintaining positive control. Different security mechanisms are implemented in different digital systems. These mechanisms range from physically locking the system to encrypting the programs and their data. However, in most cases the security mechanism can be broken and secured information can be exposed, valuable recourses abused, and mission-critical systems halted or destroyed. History has shown us that no system is fully secure and given enough time and resources any secured system can be breached.

Security breaches could originate from different sources. These could be inside attacks by a disgruntled employee or for the purpose of committing fraud. In an inside attack employee breach the security themselves or expose the system's security information to a third party. An example of a typical inside attack is a lunchtime attack that can only occur during a small window of opportunity when a system is switching between different time zones or control systems. Focused attacks are attacks that are organized by persons who are sufficiently motivated to willfully want to destroy and disrupt the normal operations of a system. Accidental attacks are attacks that use breaches that occur accidentally due to incorrect configuration or lucky guesses. Any security system has to provide good coverage against all of these different types of security breaches.

Although a security system cannot stop attackers from trying to attack the system, the security mechanisms must be designed in such a way that they prevent the attacker from gaining access to the secured information or control of the secured system. This can be achieved by anticipating the actions of an attacker. After breaching a secured system an attacker might want to eavesdrop on communication and messages; sabotage data by modifying or change data values; cause denial of service by blocking others from communicating with the system; carry out a man-in-the-middle attack, or even take over all or part of the system. By properly implementing well-designed security policies and mechanisms one can attain the desired level of security[19].

Security policies are a set of rules that govern what security decision must realize. One form of security policies concerns the assignment, management, and enforcement of access rights. A basic means of implementing security policy for access rights is an access control list (ACL). An ACL contains a list of devices and services allowed to access the devices (perhaps specifying even the specific way each actor can access the resource). When a service is requested by the device or other actor the ACL is checked for an entry for this device (based upon the name or address of the device), if a match is found then the service request is granted based upon the application of the corresponding rule in the ACL, otherwise the service is denied. ACL based security depends on the address or name of the device or other actor being true & accurate **and** that the integrity of the ACL is maintained, if either one is compromised, then security will be breached. Additionally, use of an ACL does not provide immunity against an eavesdropping attack or a denial of service attack. In fact, attacks against the ACL can be used to mount a denial of service attack. Advanced security polices utilize cryptography and authentication of nodes by using public key certificates.

3.1 Cryptography

Cryptography is the science of encoding data so that someone *without* the secret key cannot easily derive the coded information[19]. Cryptography has been used to keep information secure since the time of ancient Egypt. Encoding machines ranging from the Enigma machine built by the Germans in the WWII to the latest quantum cryptographic machine have been used to encode and decode sensitive information. For the encrypted data to be decrypted the receiver must have the secret key and a suitable machine (or algorithm), which can use the key in order to decrypt the message.

The primary shortcoming with simple symmetric key cryptography for communication is that all the parties involved in the communication must know the secret key to decrypt the message. An alternative solution proposed by Diffie and Hellman, is to use a public-private key mechanism where everyone knows the public keys of each participant, but keep his own private key secret. In order to decrypt any message from a node the receiver uses the public key of the sender and its own private key to sign the resulting message. The receiver uses its private key to decrypt the message and the sender's public key to authenticate the signature on the message. An attacker would need to calculate the private key from the public key, which is very challenging[20, 21].

More formally the sender publishes their public key Y_i where $Y_i = a^X \bmod q$. For a known a and q and a randomly selected secret key X.

For node J to get the key it uses the public key Y_i and use its private key X_j to calculate: $k_{ij} = (Y_i)^{Xj} \bmod q = (a^{Xi})^{Xj} \bmod q$.

While an attacker has to calculate $k_{ij} = Yi^{(\log aYj)} \bmod q$.

In practice public-private key encryption provides a very reliable means of data security. For example, if Bob and Alice want to communicate Bob publish his pubic key in secured places were Alice could access it. Then Alice can use this key to encrypt her message and sends the encrypted message to Bob. In this scenario as only Bob knows the secret key, he is the only one who can decrypt the message; theoretically not even Alice could decrypt the encrypted message. Bob use his secret key to decrypt the message, thus Alice's communication to Bob is secured. A similar scheme is used in the reverse direction to secure Bob's communication with Alice.

The advantage of this technique is that if the data is tempered with, then the receiver cannot decrypt it. This provides additional protection against data tempering. The shortcoming of this technique is it relies heavily on the authenticity of public key. If the public key is tempered with or provided by an unreliable source (for example one who has assumed another user's identity), then the provider of the public key can listen to the communication as it can decrypt the message using its private key and can forward it to the intended party after encrypting with the correct public key[1]. Additionally, an attacker can record a previous exchange of messages and resend earlier messages and receiver would accept them as valid messages. To overcome these problems different authentication mechanisms have been integrated into public-private key security systems to better secure the communication between the intended parties.

[1] Note that if the intended receiver knows the sender's correct public key, then they can detect that the message is not authentic. This means that the attacker has to prevent the intended recipient and the sender from getting the other's true public key.

3.2 Authentication Certificate

As it is in real life, one is authenticated by his/ her signature and identification card (such as an ID card or passport), A potential solution for authentication is to use a digital signature and a certificate. Digital signatures provide authenticity, integrity, and assert that some data is genuine. A digital signature is a mathematical scheme that passes a block of data through a function, such as a hash function, to generate a fixed length sequence of bytes (called a message digest) that **represents** the original block of data. By encrypting the message digest with the sender's private key and encrypting the message with the receiver's public key the receiver can decrypt the message using the receiver's private key and pass it through the same hashing function. Separately the receiver decrypts the signature using the sender's public key and compares the resulting signature with the hash result that it locally computed. If the signature matches, then the data is believed to be sent from the sender associated with this public key. This can be summarized as shown in Figure 2.

A digital signature is a very important and useful mechanism for authentication and integrity checking data, but it could easily be compromised if the public key used to verify an identity was compromised. To overcome this problem we can use certificate base authentication. This form of authentication depends upon a trusted third party. Companies with reputable trust issue certificates. These companies are called certification authorities (CA). VeriSign (later bought by Symantec) was one of the early CAs. Laws such as EU Directive 1999/93/EC regulate CAs[22].

A typical digital certificate, such as an Internet X.509 certificate, contains [23]:

Serial Number	Used to uniquely identify the certificate
Common Name	The name of the owner of the certificate
Signature Algorithm	The algorithm used to create the signature
Signature	Signature value
Issuer	The entity that verified the information and issued the certificate
Valid-From	The date the certificate is first valid from
Valid-To	The expiration date of this certificate
Key-Usage	Purpose of the public key
Public Key	The public key
Thumbprint Algorithm	The algorithm used to hash the public key
Thumbprint	The hash itself, used as an abbreviated form of the public key

Using the information contained in the certificate the receiver can check if the sender is reliable by checking first the hash of the certificate. If this hash fails to match and the name of the sender does not match the common name, then the receiver is notified of their being an error. Date entries are checked to see if the date of use is within the valid range. If all of the earlier tests are passed, then the public key is used to decrypt the message's signature. Note that the certificate message can be sent to the CA to check if sender is who it says it is or if this certificate has been revoked.

Once individual nodes can establish communication and trust, the next logical step is creating a group of trusted nodes that can freely and reliably exchange data. One common solution uses a public key infrastructure (PKI) where hierarchies of trusted certificates are created based on a reliable root certificate. Root certificates are public keys distributed by a reliable source in order to create a trust anchor. Using the public key in the root certificate users communicate with CA to create a trust. There are several CA distribution mechanisms one is with single CA trusted by all and signs for everyone such mechanism has shortcoming that it relies heavily on the root certificate and if the root is compromised all will be lost. Another mechanism is using multiple CA and several certificates are distributed to user and user can chose which one to trust, though all CAs are equally trusted. Another mechanism is chains of trust were one CA could delegate another user by delegating another user who will sign other certificates on its behalf. Another mechanism is a web of trust were everyone signs the others certificate building web of trust.

Currently there are more than 50 CAs signing certificates for website securities. Most browsers usually come with all these certificates embedded in their code [24]. Even so most of the market share is dominated by few of the CAs in the list. A survey done by [25] show not only few CA dominate the market but also most popular websites use certificates signed by the top CAs. The survey show more than 90% of the SSL certificate (Secured Socket layer certificate) is issued by four companies, namely Symantec Group (which owns VeriSign, Thawte, and Geotrust), Comodo, Go Daddy Group, and GlobalSign. And most popular site like Facebook.com, Google.com, Yahoo.com, and Amazon.com use certificates from Symantec Group.

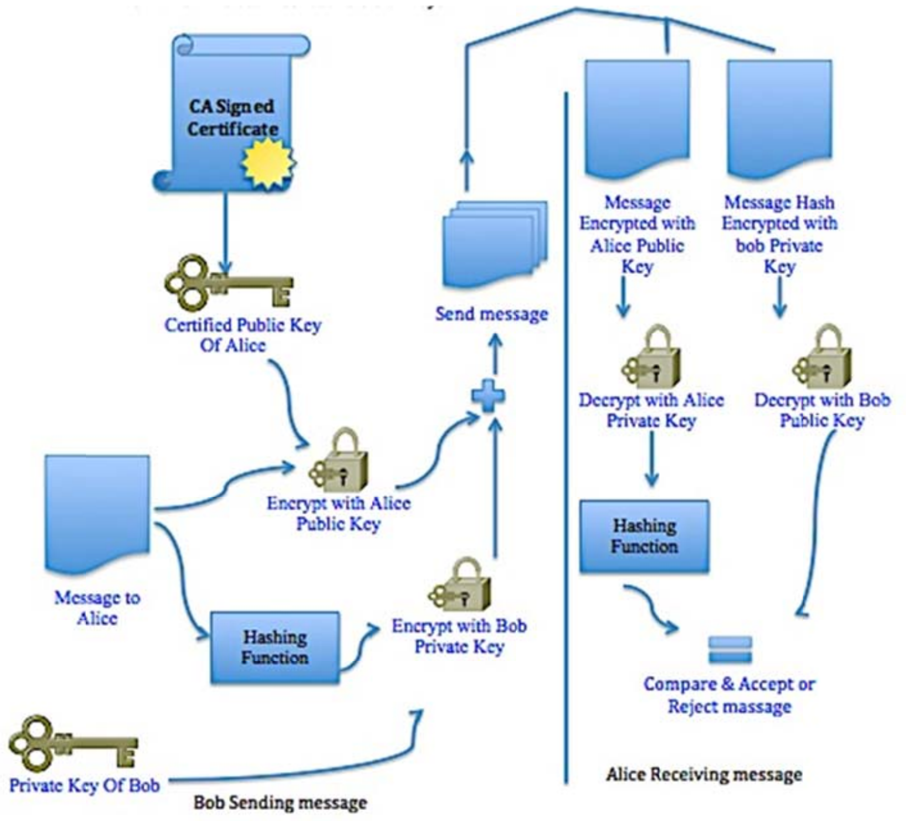

CA Signed Certificate
Certified Public Key Of Alice
Message to Alice
Encrypt with Alice Public Key
Hashing Function
Encrypt with Bob Private Key
Private Key Of Bob
Bob Sending message
Send message
Message Encrypted with Alice Public Key
Message Hash Encrypted with bob Private Key
Decrypt with Alice Private Key
Decrypt with Bob Public Key
Hashing Function
Compare & Accept or Reject massage
Alice Receiving message

Figure 2: Public Private Key encrypted communication

3.3 Trusted Computing

Ascertaining identity is an important factor in computer security. Certificates and secret keys embedded in software provide some sort of identity and security, yet they are not without faults or vulnerabilities[26]. The United States of America Department of Defense (DoD) defined a trusted computer system as one that would “employ sufficient hardware and software integrity measures to allow its use in processing multiple levels of classified or sensitive information”[27]. In trusted computing identity and security are asserted to a higher degree and trust is established among all involved in any communication. That is to say that all parties are who they say they are **and** their identity can be attested to at any time. Special hardware called a Trusted Platform Module (TPM) is frequently used to achieve this.

In October 1999, the first TPM specifications were developed by the trusted computing platform alliance (TCPA), which included HP, IBM, Intel, Microsoft, and others. These specifications become an open industry standard in 2002. In April 2003, the TCPA was suspended and a new group called the Trusted Computing Group (TCG) adopted the TCPA specifications and shifted their focus to expansion of the use of the specification and maintaining its further development. Details of the different working groups and published TPM specification can be found at the trustedcomputing.org website. The number of TPM equipped devices is ever growing, with 50 million TPM equipped devices in 2006 and 250 million devices in 2010; and the number of devices is expected to grow even more in the coming year[27].

TPM are dedicated hardware modules optimized to provide security and safeguard private and secret data. To achieve this any TPM must have the following capabilities[26]:

- Protection capabilities and a shielded location to protect all the secrets in the TPM from interference and prying.
- Platform attestation, which provides a proof of a TPM secret and the TPM’s credentials and identity.
- Integrity measurement, storage, and reporting, where platform characteristic are measured and any untrustworthiness is immediately reported and logged (i.e., a record of this stored).

The fundamental services expected in a TPM in order to fulfill the specifications outlined above are embedded in the following three roots of trust: Root of Trust for Measurement (RTM), Root of Trust for Storage (RTS), and Root of Trust for Reporting (RTR). With these roots of trust the platform can securely protect keys, evaluate (measure) stored software and data, and report on the working and integrity of the TPM. Together these form the basic root of trust on which all other trust chains are built[26].

RTM is computing engine which performs integrity measurement on different fundamental systems, *including itself*. The code to be checked could be stored in the TPM or in the BIOS boot block. This forms the Core Root of Trust for Measurement. At every boot the RTM performs a measurement of the TPM, BIOS, Master Boot Record, and OS in sequence and the current measurement is compared with the corresponding previous measurement in a Platform Configuration Register. Any disagreement is reported and system bootstrapping is halted.

The RTS provides secured and isolated storage for storing secret keys and measurements. Additionally, the RTS provides confidentiality and integrity to the data used by the TPM but stored externally. The RTS also implements data binding and data sealing by encrypting data with different keys. The RTR provides reports on shielded locations and attests to the authenticity of stored data with an Attestation Identity Key (AIK) when challenged. Together the RTS and RTR provide the basic service expected from a TPM by using different components, such as non-volatile-memory, a SHA-1 hash engine, a random number generator, a key generator, and an input/output bus controller.

Although the TPMs provide long sought security and key protection, they are not without limitations or shortcomings. TPM are designed to provide protection against software attacks and software hacks, but they remain vulnerable to physical attacks. In [28], J. Winter and K. Dietrich (presented at the 8th European workshop on Public Key Infrastructure, Service and, Application conference in Belgium) their results showing that at a cost of less than 100 Euro and enough knowledge that one can successfully hack in to TPMs and compromise the root trust. In addition to this, shortcomings regarding privacy and spying; remote censorship; loss of control of a personal machine because keys are managed by a supplier; and issues of using open source software and how to sign and authenticate such software remain major concerns [29, 30].

3.3.1 TCG TPM Specifications

The TPM specification as proposed by the TCG and specified in the TPM Main Part 1 Design Principles Specification Version 1.2[31] is partly summarized and presented below.

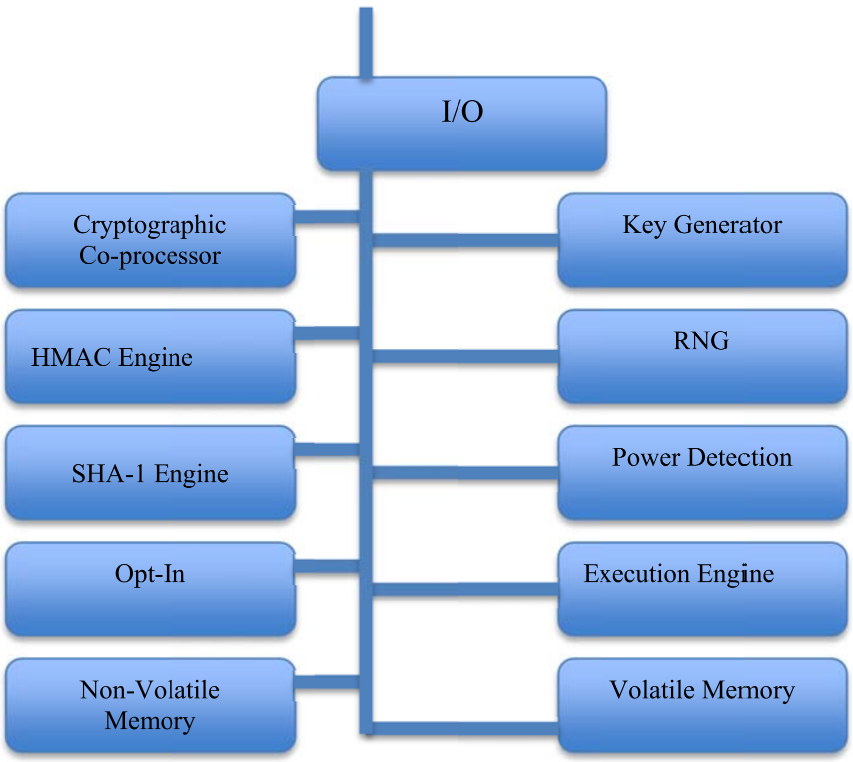
Interoperability TPM must support the RSA, SHA-1, and HMAC algorithms

Components: TPM must have the major components shown in **Error! Reference source not found.**.

- The **I/O module** manages information flow over the communication bus and implements access control.
- **Cryptographic co-processor** facilitates asymmetric key generation (using the RSA module), asymmetric encryption/decryption (using the RSA module), cryptographic hashing (using the SHA-1 module), symmetric encryption (AES), and random number generation (using the RNG module).
    - RSA module/engine provides digital signature encryption using the RSA asymmetric algorithm.
    - Signatures can be computed for internal items and on request for external items.
    - A symmetric encryption engine (possibly using AES or XORing with one-time pad) is used to encrypt authentication information, which provides confidentiality in transport sessions and provides internal encryption for data stored outside of the TPM[2].
    - Symmetric or asymmetric keys a generated using the RNG and their integrity is protect with a BIND and SEAL capability.

[2] Though here the symmetric encryption engine is depicted as part of the cryptographic co-processor it could also be implemented as a separate module.

- **Key generator**: generates symmetric key, RSA - asymmetric keys, and nonces. Asymmetric keys must be generated in a protected manner so that they can be used as private keys.
- **HMAC Engine**: calculate message hash to verify request received from processor are authorized and they haven't been tampered on the way.
- **Random Number Generator** (RNG): generates random numbers for nonce, key generation, and random signatures. Within the RNG are the following subsystems:
  - **Entropy source**: must provide as much unpredictable data as need.
  - **State register**: hold the current RNG state even during power-down.
  - **Mixing function**: takes the state register values and generates a random number.
- **SHA-1 engine**: for hash implementation of measurements during boot phase.
- **Power Detection**: manage TPM power state and platform power state to support physical presence attestation and restrict some applications from running when system is physically constrained.
- **Opt-in**: used to enable/disable, turn on/off, and activate/ deactivate the TPM. This can be done either with authorization from the TPM owner or given physical presence at the platform.
- **Execution engine**: runs program code to execute the TPM commands received via the I/O port.
- **Non-volatile memory**: stores encryption key and states associated with TPM. Platform Configuration Register (PCR), which is a SHA-1 digest of all measurements needed for secured booting, and Data Integrity Registers (DIR), which is a digest summary of PCR values, are the states associated with TPM.



Endorsement Key Creation: An endorsement key (EK) is burned into the platform and signed by the platform creator. The private portion of this key never leaves the TPM, while the public portion is accessible.

Attestation Identity key (AIK)**:** The AIK is asymmetric RSA key used for signing data generated by the TPM. The AIK is an alias for the EK and certificate authorities are used to attest to the validity of the AIK public key.

These are the major elements of the TPM specification that should be incorporated into any system to achieve trusted computing. More detail could be found on the TCG website [26,31]. The next sections will evaluate three TPM implementations; this should help establish comparison between the specification presented here and actual implementation. Microsoft, ARM, and the hardened computing described in the book edited by Chris Mitchell have done these three implementations.

3.4 Microsoft's Next Generation Secure Computing Base (NGSCB)

Microsoft was one of the founding member of the TCG and an active member in developing the TCG specification Version 1.1. In 2002, Microsoft announced their plan to integrate a system, which combines hardware and software [26]. The system initially was named Palladium, later renamed Next Generation Secure Computing Base (NGSCB). It combines changes made by hardware manufacturers, a Secured Support Component (SSC), and a minimal isolation kernel to implement TPM. In addition to the TCG functionality, NGSCB extends to the CPU support for a minimal kernel, a minimal isolation kernel, an extended chip to control DMA, and to secure hardware based I/O.

The SSC is a tamper-resistant cryptographic chip implementing most of the functionalities specified by TCG. It implements a cryptosystem, a RNG, small memory for counters, and PCR. With this component and the isolation minimum kernel the SSC is able to Seal/Unseal data and keys; get the Entropy of the RNG; and manage (read or increment) monotonic counters. These capabilities enable the NGSCB to fulfill most TCG requirements[26].

The isolation kernel utilizes modifications made by the hardware manufacturer to execute in the CPU with higher privilege than any other OS(s). The isolation kernel runs in mode -1 (while other OSs run at mode 0) this gives the advantages of supporting both virtual machines and exo-kernels[32]. Additionally, a chip extension is used to implements DMA access control. The chip contains a policy map indicating who gets to access which region of memory and when, and this policy is enforced using hardware. The result if that memory is partitioned for guests OSs and the isolating kernel.

3.5 ARM TrustZone

The need for secure trusted computing platform security features in embedded systems, such as smart phones and PDAs, is being tackled by TCG for mobile phones and TCG for embedded system. ARM TrustedZone is security technology for embedded system that has focused on the issues that matter most in embedded systems, such as power, area, and performance. The TrustedZone implements an isolated secured environment which allow a software implementation of high level security services, such as TPM[26].

There are number of ways to secure embedded systems, some are complete hardware security systems, which are based upon an inflexible, off-chip co-processor. Such as co-processor can easily be compromised by tapping the communication line between the co-processor and the main core[28]. Another variation of this is SIM card based security, which suffer from both memory and bandwidth limitations. ARM saw these shortcomings and implemented the TrustZone with two parallel execution worlds: the non-secured normal environment and a trusted certifiable secured world[33].

Major components in the trusted zone are the TrustZone CPU (used to run trusted application in the trusted zone); secured on-chip boot ROM to configure the system; on-chip non-volatile one time programmable memory to store master keys; secured on-chip RAM used to store encryption keys; and other resources to manage peripherals and allow access only by trusted applications. Developers can use these components and the TrustZone APIs to get full access to the trusted hardware and to implement TPM or even better security.

3.6 Hardened Processors

The “hardened processor” as referred in [26] is based on the Executable Only Memory implementation proposed by [34]. This implementation takes security to a higher level, by not only integrating the security module on chip but also by encrypting all communications in and out of the CPU. The only reliable (and trusted) information is that found in the executable only memory and all communication is encrypted using a session key as a symmetric key and public private encryption using a key embedded in the chip. Session keys are used to encrypt data using fast symmetric encryption algorithms and each storage tag its data with some session key identifier tag. These tags are mapped to session keys using a session key table in the executable only memory.

The executable only memory uses hashing and a mutating session key table to protect the system from spoofing, splitting, and replay attacks. The session key table associates a mutating register which is changed at every interrupt, thus if a command is replayed the deciphering table will invalidate it because its identifier will not match any entry in the table. This mutating table protects the system from a replay attack.

4 SCADA system: Networks and Protocols

Communication and networking are key elements of the infrastructure of a supervisory and control system. Establishing fast, reliable, and easily understandable communication relies on the protocols used for network communication. For these reasons SCADA system designers have developed a number of different networks and network protocols to suite their design requirements. Unlike the network protocols one is accustomed to, SCADA protocols are simple, lightweight, and designed to provide fast communication in a very isolated industrial environment. Commonly supported network architectures are hierarchical networks; mesh networks; or master/slave point-to-point networks running over serial links.

The advances in internetworking technologies have forced these simple protocols to undergo major changes. In order to support larger numbers of nodes, the SCADA protocol designs must address security, expansion, and modifications in order to internetwork with corporate networks. The following sections briefly present some of the well-known SCADA protocols in more detail. Further information can be found in the references cited in each section.

4.1 Modbus

Modbus is a transmission protocol first developed by Gould Modicon (now Schneider) in 1979 for a process control system. Since it was designed *without* a specific interface it has been easy to adapt to different networks, each with different interfaces. Modbus is the *de facto* factory standard and is utilized in more than 40% of industrial interconnections. Originally, Modbus was designed to run with a master and slave paradigm allowing up to 247 slaves, but variants such as Modbus TCP/IP and Modbus plus support for more nodes and all of these nodes can initiate and participate in communication[17,35]. All Modbus protocols come in two variants: an ASCII version (which is slower, but printable) and an RTU version with a binary (hexadecimal) format for fast communication. However, the binary version is unprintable[3].

A typical frame has a 1-byte address field, a 1-byte function field, a variable length data field, and a 2-byte error check field. The ASCII format uses a colon (:) as a header to separate each frame. In contrast, the RTU format use 3 blank signal cycles as a separator thus if 3 cycles pass without a signal the next byte received is the address of the slave. Each slave has a unique address and all slaves on the network listen to the master's messages and only the slave with the specific address in the request will respond. The function field of the frame tells the slave what actions to take and how long the data field is. In the error check field the RTU frame contains a 16 byte CRC checksum, while the ASCII encoding uses a longitudinal redundancy check (LRC)[35].

[3] However, it can be dumped in hexadecimal and other formats. Additionally, it can be decoded by various tools, such as Wireshark.

4.2 PROFIBUS

PROFIBUS is a smart field bus technology initially designed by the German government and German automation industries and now owned by PROFIBUS & PROFINET International (PI). PROFIBUS specifies a three-layer implementation of the International Organization for Standardization (ISO) Open Systems Interconnection (OSI) model providing reliable communication, self-diagnosis, and connection-diagnosis. The application layer provide applications, such as DP-V0, for cyclic data exchange; DP-V1, for acyclic and alarming handling; and DP_V2, for broadcast and slave-to-slave communication. The data link layer defines a slave-master connection service and a “token” type connection. The physical layer defines standards based upon RS-485, a communication standard designed to run over twisted pairs and (in some variants of PROFIBUS) over optical fibers.

4.3 PROFINET

PROFINET is an open standard developed on top of the TCP/IP standard to provide real-time reliable industrial Ethernet based communication system. PROFINET provides high-speed operation, seamless operation, and support for time critical motion control. A typical PROFINET network contains an IO-Supervisor, typically a HMI or PC; an IO-controller, typically a PLC used to communicate with sensors and motors IO-devices (which are mostly motors, sensors, pumps, and valves); and of course the sensors and motor IP-devices themselves. In the network the IO-controller maps the IO-supervisor’s commands to the appropriate IO-devices. Network services provides a cyclic data exchange, between IO-Devices and the IO-Controller, As with any cyclic data exchange protocol, these devices exchange configuration and diagnosis information, alarms, and the results of message processing.

4.4 High Level Data Link Control (HDLC)

HDLC is a predecessor of Ethernet defined by ISO for use in point-to-point and point-to-multi-point communication. HDLC can run in unbalanced normal response mode (NRM), which has one master initiating communication, or in asynchronous balanced mode (ABM), where all nodes can initiate communication[17]. There are three frame formats: unnumbered frames used for establishing a connection, information frame sequence numbered data frames, and supervisory frames sequence numbered frames used for flow control and error messaging.

A typical frame contains a start of frame delimiter flag that is eight bits; an address field of eight bits to indicate the receiver node, group, or broadcast address; an eight bit control field to indicate different frame formats and exchange status; sixteen bits for frame check sequence number; and eight more bits for a frame delimiter. If there are more nodes in the network, then the address field can be extended[17].

The primary node (i.e. the sender) initiates communication by sending unnumbered frames; this is interpreted as a request for a connection. The secondary node (i.e., the receiver) responds by sending an unnumbered frame indicating its status. If the receiver has data ready, then this data is transferred with a data frame and when transfer has been completed the primary node sends an unnumbered frame requesting that the receiver disconnect and the secondary node acknowledges this request with an unnumbered frame. Data exchange is monitored and sequenced by the exchange of supervisory frames.

4.5 Distributed Network Protocol (DNP3)

The International Electromechanical Commission (IEC) proposed a 3-layer implementation of the OSI model and specified several different frame formats. Harris Controls developed DNP using the IEC 870-5-1 FT3 frame format, by modifying the three layers to four, and IEC developed IEC 60870-5-101. DNP has significant acceptance in America and Asia while IEC 60870-5-101 has wider acceptance in Europe[17]. Both protocols, DNP and IEC 60870-5-101, are open communication protocols.

The 4-layers of DNP are physical layer, data link layer, pseudo-transport layer, and application layer. The application layer organizes data into an application service data unit (ASDU) with a maximum size of 2048 bytes and passes this ASDU to the pseudo-transport layer. Here the ASDU is fragmented into packets of 249-bytes and a 1-byte header is added forming a transport service data unit (TSDU). TSDUs are encapsulated with a 10 byte data link layer header and 16-bit CRC trailer to form a link protocol data unit (LPDU). The LPDU is bit streamed using RS-232C voltage levels and control signals by the physical layer.

4.6 TCP/IP

TCP/IP is the de-facto standard for network and transport layer implementation of interconnections. TCP/IP was the first proposed internetworking protocol and is based on the four-layer TCP/IP stack. TCP/IP introduces an Internet protocol (IP) for interconnection and applications make use of a transport protocol. Applications can use the transport control protocol (TCP) to ensure reliable in order byte stream host-to-host communication. The four layers of this protocol stack are: application layer which identifies the application that created the data; transport layer for host-to-host communication with potentially flow control and error detection; internet layer for datagram exchange across networks using internet addressing; link layer which handles the data to electrical (or optical) signal conversion and framing; and a physical connection - commonly using Ethernet’s many different physical layers. TCP provides reliable communication between different physical networks. The user datagram protocol (UDP) can be used to provide low overhead unreliable datagrams over an Internet.

5 Assuring Industrial Control System Security

This chapter will try to answer the following two questions: "What are the security risks associated with SCADA systems? What are current best practices?" Then the chapter ends with a presentation of a number of systems and solution that implement these best practices.

5.1 SCADA Security

Industrial control systems, be it a SCADA or distributed control system (DCS), typically control a mission critical process, thus any security breach may result in havoc, destruction, or service interruption. Unlike other cyber security breaches, which may result in a loss of personal information or money, a security breach of a SCADA network can cause damages ranging from introducing a flaw in the products to a complete plant shut down or loss of life and environmental damage. With the increasing political and social instability in the world threats against SCADA systems are increasing every year[36]. As a result the need to ensure the security of SCADA systems and protecting them from exploits has become a national security matter[37].

5.1.1 Security Analysis of SCADA

SCADA systems were designed for high availability and reliability as expected by large industries and utilities. The major design requirements are safety, reliability, efficiency, durability (i.e. surviving in a harsh industrial environment), and constant availability (i.e., around the clock and throughout the year). These systems were traditionally designed to work in ***isolated*** industrial environments using custom made interfaces and protocols running over serial links. Therefore the need for security was seen as being limited to ***physical security***. However, with the advances offered by the Internet and the cost savings offered by interconnecting this isolated control system to other networks, the benefits were believed to outweigh the imposed security risks [38,39]. But the lack of robust security feature such as encryption and the fact they control critical infrastructures has made SCADA systems and ideal target for attack.

Security analysis, tests, and countermeasures in SCADA systems are very complex and special consideration is necessary. For example, patching a SCADA system in the same way as is commonly done for PC software would require a system restart; however, this is almost impossible because a system restart would require shutting an entire plant down, removing all the waste caught in the system, and restarting it. In the case of a power plant a shutdown might black out an entire region. Implementing encryption and introducing complicated network filtering rules could delay communication via the network causing a critical system failure[39]. So finding appropriate methods to test and performing security analysis on an operating SCADA network becomes important, as we cannot simply blindly apply the methods used with non-critical systems.

Unfortunately, there is no cure-all solution to SCADA security problems, i.e., a solution that would solve all the security problems. As a result security risk analysis and testing must be carried out on individual plants to come up with plant specific solutions (although it is possible to replicate the solutions as one replicates instances of a specific plant design). Munro [38] and Ralston, et al. [40] provide good insights into testing and risk assessment.

5.1.2 What are the causes of security risks?

To calculating the risk associated with any security breach in a SCADA system, we have to consider the likelihood of an event and its impact, as *Risk* is a product of the *likelihood* of event and its subsequent *Impact*[1]. It is evident that the potential risk is generally high due to the type of industry that the SCADA system is serving, as the impact is generally quite large – even if the likelihood of a specific event is small without an intruder. Unfortunately, these systems are often large and complex and a loss of control can easily happen, so it is very difficult to know what caused the impact. In his book, Munro says "*Would we ever know if a power cut was a result of a normal system overload, or was the result of a hacking event?"* [38] For these reasons we can understand that the goal of an attacker is to turn a naturally unlikely event, i.e., an event which is very unlikely to happen during normal operations, into an actual event.

Byres and Lowe[36] study different known incidents concerning SCADA systems and categorize the cause of the incident into three categories. These are **internal cause** - usually due to a disgruntled employee; **accidents** - due to human or machine failure; and **external cause** – such as an attack by hackers. Their study shows that in the years 1980 to 2000 the proportions of these three causes of incidents was about the same for all three categories: external 31%, internal 38%, and accidental 31%. However, after the year 2001 the trends changed with external causes doubling to 70% while internal causes shrank to 5%; and accidental causes remained roughly constant at 25%. This shows that external causes and threats have become more significant as the cause for an incident. As a result the risk associated with external causes is increasing every year. To make matters worse the risk is also increasing because the impact of an incident is increasing due to growing plant sizes, increased plant efficiency, and consolidation of industries.

The reasons suggested for the increase in external attacks are the networking within and the external network connections to the SCADA system; the increasing use of common operating systems, which have well known vulnerabilities and are readily available to hackers for experimentation and development; and the advances in the development of worm and viruses. A study by Byers and Lowe [36], further analyzes the threat imposed from increased networking. The primary entryways for attacks are remote Internet and remote dial-up connection access, followed by VPN connections. Most of the vulnerability of SCADA systems originates from mixing the business network with SCADA network. According to Byres and Lowe this mixing of networks alone accounts for 43% of all incidents they studied, followed by faults and security breaches in the HMI that accounted for 29% of all the incidents they studied[36].

It is clear that security threats and breaches will increase every year unless the vulnerabilities due to increasing networking and weaknesses in the HMI are adequately addressed. For this reason, a thorough investigation into these vulnerabilities and possible mitigation methods is mandatory. However, in this thesis we will focus on the networking aspects of the problem.

5.1.3 Top ten SCADA vulnerabilities

The top ten common vulnerabilities of SCADA system, as presented by the North American Electrical Reliability Council are [41 ,39]:

- **Inadequate policies, procedures, and culture that govern control system security.** The policies and procedures implemented do **not** address the overall security needs or might be outdated. Additionally, most employees are unaware of these policies.
- **Inadequately designed control system networks that lack sufficient defense-in-depth mechanisms.** Most SCADA networks were initially designed for availability and reliability, thus addressing security might require redesigning or integration of additional components into the system.
- **Remote access to the control system without appropriate access control.** Two of the major causes are inappropriate use of dial-up modems and poor passwords.
- **System administration mechanisms and software used in control systems are not adequately scrutinized or maintained.** Inadequate patch management, lack of up to date virus protection, failure to remove unnecessary or obsolete accounts and removing former user's privileges, and obsolete software can expose the system to easy worm and malware attacks.
- **Use of inadequately secured wireless communication for control.** Using off-the-shelf devices, which do not fulfill industry standard quality and security requirements could expose SCADA network to easy exploitation.
- **Use of a non-dedicated communications channel for command and control and/or inappropriate use of control system network bandwidth for non-control purposes.** Internet based SCADA; using the control channel for non-control purposes; and having devices connecting to Internet from the control network could easily expose the system to a denial of service attack.
- **Insufficient application of tools to detect and report on anomalous or inappropriate activity.** Under-utilization of intrusion detection systems, network management systems, and network filtering/system isolation are common in security incidents.
- **Unauthorized or inappropriate applications or peripheral devices on control system networks.** Unauthorized peripherals, laptops, USB-memory sticks, and PDAs used with access to SCADA network.
- **Control systems command and control data not authenticated.** Using protocols that do not support authentication or no/poorly implement authentication.
- **Inadequately managed, designed, or implemented critical support infrastructure.** Insufficiently tested or maintained redundant systems; poorly protected communication or power supply lines; and inadequate fire suppression system are examples of inadequate critical supporting infrastructures.

These vulnerabilities can be summarized and categorized in five main groups as follow. Network and communication vulnerabilities, vulnerabilities due to lack of security in the communication protocol; lack of security in the communication media, such as radio and dialup; and lack of access control on the communication network. Data vulnerabilities, vulnerabilities due to data being stored or communicated unencrypted and due to bad encryption key management. Underlying OS and software vulnerabilities, vulnerabilities due to faults in the OS; due to faults in the software used to manage and program the PLCs; and due to lack of mechanism to check anomalies. Authentication and access right vulnerabilities, vulnerabilities due to poor password management; due to poor encryption key management; and due to poor access privilege policy. Policy and culture vulnerabilities, vulnerabilities due to poor security policy that does not address all security needs; due to lack of security policy awareness among workers; and due to lack of policy to regulate use of peripheral device.

Any good SCADA practices for producing a secure SCADA system must implement methods to address the above vulnerabilities.

5.2 Best Practices and recommendations for a secure SCADA system

SCADA systems are rapidly being integrated with corporate networks and the Internet, thus isolation is no longer an option. Not only is physical isolation impractical, but also increasingly it is infeasible from a business point of view as the operations of the facility are more tightly coupled to external actors (due to the desire for increased efficiency and to reduce pollution and waste). However, this increasingly open access facilitates accidental attacks. VLANs based on IEEE 802.1q using dynamic trunks can easily be exploited[38], dial-up and serial modem connections can easily be tapped, and worms and malwares create even more holes in the system thus exposing the system to farther attacks. These attacks can reprogram one or more of the network connected devices. Vulnerabilities in the operating systems of the HMI can easily be exploited to provide access to the SCADA system; all it takes is a single malicious e-mail to a SCADA administrator or field engineer. Additionally, it is often easy for an attacker to connect their own laptop to the network and have free access to any of the networked systems (if they connect to the network *inside* the firewall).

A combination of polices, rules, and security mechanisms must be utilized to achieve the desired SCADA security. Eric Luiijf has offered a few suggestions for best practice[42] in SCADA systems and these practices can also be used to evaluate suggested implementations:

- Develop security policies and regulations to address security issues throughout all the four levels of the production model.
- Outline security procedures and safety regulations to be taken in case a security breach is detected.
- Establish a good code of practice for information security at both the third and second level of the production model.
- Establish a responsible body to make sure that the security policies and security code of practice are implemented. This body should also run regular security checks and present their estimate of the *achieved* protection level.
- Schedule and run regular audits of the network, devices on the network, software running on the HMI, and access rights and privileges of both operators and applications.
- Outline the security rules and requirements expected from all devices and software before it can be allowed to join the SCADA system. Make sure that each provider adheres to these rules and regulations.

- Establish mechanisms for managing, testing, and evaluating maintenance and upgrades of software and hardware *before* they are applied to the SCADA system.
- Implement proper network segregation using firewalls and gateways.
- Implement proper minimal privilege rights and remove unused links. Filtering and monitoring should be done on all packets going to and from SCADA systems.
- Implement encryption, authentication mechanisms, and regular changing of passwords and security procedures.
- Document SCADA system processes, configurations, and modifications to help monitor anomalies and unauthorized changes.
- Establish a process database and run regular check to make sure the running process conforms to routine and schedules.
- Control what goes in and out of the SCADA system by controlling network traffic and regulating or prohibiting use of removable media.

Increased environmental security; establishing security-sharing centers; exchanging security threat information; and continuous studies on potential security threats would also help strengthen security.

5.3 Possible implementation (solutions) for SCADA security

SCADA systems are **vulnerable** and **critical**, so proper solutions must be implemented to secure these systems. Any solution must implement sufficient isolation; implement policies and regulations for protection against misuse; implement rules for encryption and key distribution; and assure reliability and accountability by logging system reports and alarms. A few selected solutions proposed by different scholars and companies are briefly described in the following paragraphs.

In [43], Vinh Ich Nguyen, Watit Benjapolakul, and Krisada Visavateeranon proposed an Internet based SCADA system using low cost embedded TCP/IP boards and a server PC. Their solution is implemented using a sever computer connected to the Ethernet network of the SCADA system. Low cost Ethernet interfaces connect the PLCs and the SCADA network with the server computer. A router and modems are used to connect the SCADA network to the Internet. Client/server applications are run both on client computers and the server computer. These applications are used to manage resources, add or remove modules, and display the status of any of the components of the SCADA network. The server computer serves as a middleman between the PLCs and an operator who controls and configures the PLCs through the Internet. The server computer runs a database containing the set of allowed commands and every command sent to the PLCs is checked against this database. And only if the command is permitted for the specific user then command is forwarded to the PLC. Additionally, the server computer checks the user's privileges allowing the implementation of different access levels based on password authentication.

A typical usage scenario would be, a client running a client application with the correct privilege securely sends a command to the server computer and the server computer decrypts the command and checks it against list of allowed commands for this user. If the user is permitted to issue this command, then the command is sent to the PLC and PLC takes some action. Security between the client and the server computers is implemented using password authentication, MD5 encryption, and network filtering. Additional filtering is done on the Ethernet interface to hide the IP addresses of the PLCs and to protect against direct access from the Internet.

This is a simple and low cost solution, but it suffers from some shortcomings. The security mechanism is distributed among different modules of the system (the server, network routers, and Ethernet interface), thus for an attacker to find loopholes would not be difficult. The system relies heavily on the server computer making it a common-point failure and any breach of the server computer would cripple the system or might even put the plant out of operation (or even control). Simply checking each command against a list of allowed commands does not guarantee security as combinations of allowed commands could cause fatal damage to the plant. Nonetheless, their solution proposes a system module management and monitoring mechanism.

In [44], I. Nai Fovino, A. Carcano, and M. Masera propose a secure and survivable SCADA system utilizing the Modbus protocol. Their solution is implemented using multiple level signed communications, a SCADA firewall, and a reliability voting mechanism. The system uses Modbus with a limited packet size of 256 bytes and timestamps each packet to prevent a replay attack. The master HMI is connected to multiple hardware modules, each of which is called a filtering unit. These filtering units are in turn connected to the PLCs. Communication between the master HMI and the filtering unit, as well as communication between the filtering unit and PLCs, is always encrypted. Additionally, multiple filtering units are used and voting is done to make sure that the filtering units are not compromised.

The filtering unit implements a rule database, which determines which user, can issue which command. The filtering unit uses the system description, an event tracker, and a Modbus analyzer engine to generate an overall picture of what is going on in the network and to filter out malicious activities. The system provides protection against unauthorized command execution, man-in-the-middle-attacks, replay attacks, and malicious packets from a corrupt master while maintaining delay within an acceptable range.

This is a very robust solution, but suffers from high cost of multiple filtering units and key management and distribution might be difficult especially if the PLC or RTU are distributed over a wide geographic region (as might occur in the case of smart grids and meters). Additionally, voter reliability, and computational cost of drawing the whole network inside the filtering units are other disadvantages of this system.

In [45], S. Bagaria, B.Prabhakar, and Z. Saquib propose a mechanism to secure existing SCADA networks running the DNP3 protocol. Their solution is implemented with DNP3 version DNP3sec and a bump in the wire (BITW) hardware module, which can be directly integrated into an existing SCADA network. Their solution uses fixed size 256 byte DNP3 frames. In an ideal scenario the BITW modules will be connected to each HMI and PLC, so that data is always be routed between pairs of BITWs. The BITW module connected to the HMI takes apart the plain text DNP3 message and encrypts the payload and places this encrypted payload into a new DNP3sec frame and forwards it toward the destination. Just before the destination the BITW connected to the PLC receives the DNP3sec frame message, decrypts the message, and re-encapsulate it as DNP3 plaintext and passes it to the attached PLC.

Using lightweight encryption and efficient key exchange the solution was able to achieve the needed security and meet the bounded delay requirement. This solution is especially useful if the SCADA system is distributed over a large geographic region and running over private or leased lines (as might be done in a smart grid and meter system) or when the SCADA system uses lines leased from telecommunication companies. However, the system does not address the security risks from a compromised HMI or security threats due to malicious software. There is no access privilege based command filtering or flagging of unauthorized commands and malicious activities. This solution simply secures the DNP3 traffic between two BITWs.

In [46], L.Katzir and I. Schwartzman propose a unique way of securely updating nodes in a SCADA system used in the electrical grid with smart meter IEDs. Their solution uses the unique behavior of electrical grids and the grid frequency. They suggest that traditional key based encryption and authentication mechanisms are inefficient and vulnerable, so it is better to use a known frequency pattern on the grid for authentication. A frequency detector is attached to each smart meter and could be used to detect specific frequency patterns and if a specific pattern is detected, then the system firmware could be updated.

Their system was proposed as an efficient update mechanism for fast recovery after major incidents and all systems could be automatically updated with no loss of time as would be required for an authentication exchange. They propose a mechanism to generate the triggering frequency without affecting the grid's frequency and the attached frequency detector could easily detect the triggering. When choosing a triggering frequency, the chosen frequency must be unique and hard to generate externally. Unfortunately, the system suffers from lots of security issues, but it presents a unique solution to the authentication problem using a case specific approach.

In [47], Todd Mander, Richard Chenng, and Farhad Nabhani proposed using rule based security to block unauthorized access to a SCADA network running DNP3. Their solution proposes classifying the stakeholders and implementing restriction rules based on privilege. Restriction rules include functional code restriction; data object restriction, and temporal constraints. Functional code restrictions limit different privilege users from engaging in various activities such as read/write operations. Data object restriction prevents users from creating some data types, while allowing them to create other data types. Temporal constraints define restrictions based on circumstances such as no data access from the PLC at a specific time. The security rules are evaluated in sequence starting with identifying the user's privilege(s), checking functional code rules, followed by data object rule checking, and finally temporal rules.

Applying these rules with properly designed network security can lower or even eliminate the vulnerability of SCADA networks. The major drawback of such a system is that a compromised primary master could endanger the whole system. Additionally, security rules could become too complex to handle as the networks grow and the delay due to processing them could compromise the performance of the system. Combining this solution with the solution proposed by A. Mahboob and J. Zubairi [39] which involves several layers of firewall, DMZ, intrusion detection, system segregation, and continuous network monitoring for network anomalies could help reduce the size of the set of rules.

A service-based mechanism to secure SCADA networks was proposed by Ray Hunt and Jill Slay [48]. Their solution proposes securing the SCADA network with the help of digital forensic services, along with the standard security mechanisms. Their solution implements security using forensic tools that monitor the network by running intrusion detection and malicious activity filtering; using tools that analyze the network for threats and vulnerability; and using tools that manage and report security events. Using these techniques their solution strives to achieve real-time forensically sound traffic monitoring, logging, and alerting.

Norman AS's Norman SCADA Protection (NSP) system proposes network-based protections that prevent any malware from executing any feasible attack. They intend to do this by using a hardware module, which is placed in between the SCADA network and the corporate network, and software running on all PCs and HMI for managing thumb drives and providing protection against malware over the air attacks. The hardware module called Norman Network Protection (NNP) runs real time malware scans, blocks suspicious links, and filters packets based on protocol and MAC address. The software called Norman Device Control is run on all PCs and HMI and is used to securely manage thumb drives so that no malware could attach itself when the thumb drives are moved from computer to computer. Endpoint Norman is run on PCs to provide protection against the latest malware, viruses, and spyware[49].

An application level solution proposed by Sidney Valentine and Csilla Farkas [50] creates a database of PLC code and checks the code against a predefined set of rules. This is useful to check for code anomalies such as one to many mapping and other parametric violations. More work can be found at [51-54].

6 Method to achieve the project's goal

The overall goal of this thesis project, as mentioned in the problem statement in section 1.2on page 4 is to provide a secure means to reprogram devices connected to a SCADA network.

6.1 Detailed Goals and Requirements

The overall goal is broken down into the following sub goals:

- Assuring secured communication in a SCADA network by implementing TCM's specification in PLCs.
- Assuring code and update reliability in SCADA system using signature and authentication.
- Mitigating changes by malicious code by means of an Access Controller and state measurement.
- Improving system reliability by means of software redundancy.
- Inter-operability with other machines running TPM.
- Security that will not degrade the operation's timing, system performance, and system resources (such as the power and area of integrated circuits used to realize the system elements).

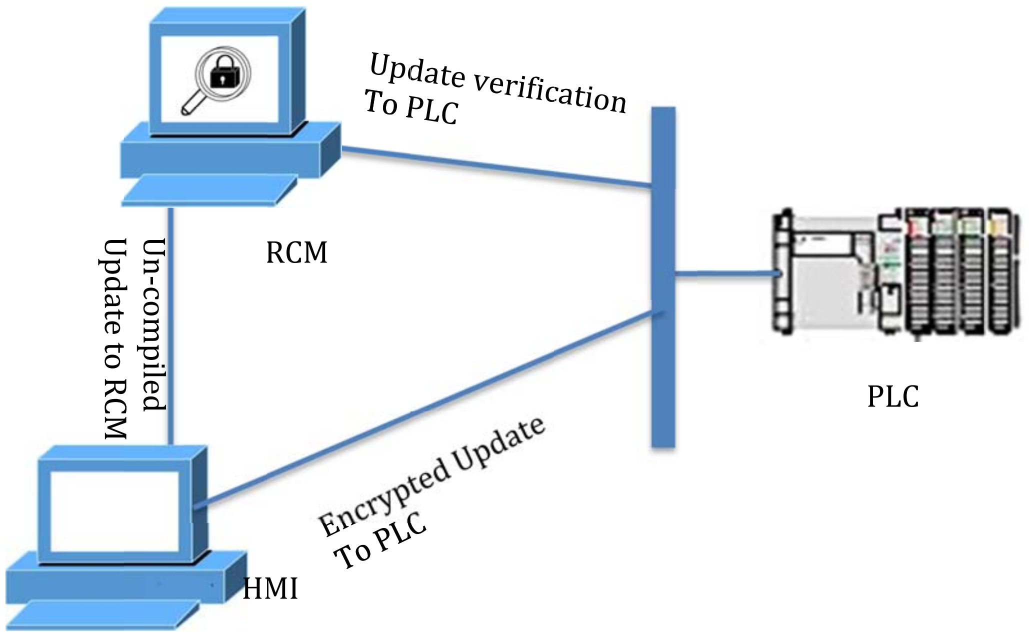
6.2 Method

The need to secure SCADA networks and the state of the art solutions that have been proposed and implemented have been covered in the previous chapters. This chapter and in the subsequent chapters will present the solutions proposed in this thesis project; explain why this solution has been chosen; and how this solution compares with previous solutions. A generalized summary of the solution proposed by this thesis is presented below.

As described in the previous chapters the vulnerabilities in SCADA system come from either vulnerabilities in the underlying OS or vulnerabilities in the hardware (inabilities of the PLCs to support and implement efficient secure network). To address these problems we propose both hardware and software based solutions. The hardware solution will provide data security and integrity on the PLCs, while the software solution will provide code reliability and provide a reliable computing machine, which can be used as reference. An overview of the system is shown in Figure 4.

The hardware solution is based on the principles of TCM, where a lightweight version of TPM is implemented and used to provide security and key management functions. The major required functionalities are: secure communication; secure key management; secure memory protection; secret key generation; and an attestation service. The TPM generates keys for encrypted communication with other modules on the network. Additionally, the TPM performs tests and measurements to make sure that the secret keys are **not** compromised and that the operating system is **not** changed at runtime. This ensure that the OS can only be changed through secure communication with a module having the correct key, thus preventing changes by malicious applications or applications that try to change the OS at runtime by directly accessing the OS's memory resident code or data.

TCM are used to realize secure communication across the network, this helps to overcome the vulnerabilities in the PLCs themselves. This leaves us to consider the vulnerabilities due to the underlying OS and application software (specifically the control software). The proposed software solution tries to address this problem by using a Reliable Computing Machine (RCM), which is used to compare and confirm that the code sent to the PLCs is the same code written by the privileged user. Since the RCM is never directly connected to the Internet or accessed directly using removable media (CD or USB sticks) the probability of it being compromised is assumed to be very low. The RCM can serve as a certificate authority that can verify the code's integrity and attesting to the validity of an update. In the worst-case scenario where an infrastructure system is compromised, the attacker would have to be able to compromise both the HMI and the RCM. Making proper use of both of these elements reduces the probability of compromise in a multiplicative fashion[4]. In addition to certifying code the RCM could maintain a database of update codes and generate a report whenever there is a change in trends.



Although many [29, 55, 56] argue that implementation of TCM based on the TCG specification has many shortcomings associated with the RSA and SHA-1 cryptographic techniques, privacy, and computational freedom; others [57-60] prefer to look the other way by saying that there is great advantage and security in the TCM specification and that the shortcomings are minor compared to these advantages. With regards to the shortcomings of the cryptographic techniques this solution tends to look the other way by arguing that their shortcomings are insignificant due to the working environment of the devices. The resources needed to break the cryptographic technique are inaccessible in the working environment, which is isolated and because of the low speed serial network that is generally used to connect the devices. The

[4] The probability of a compromise is reduced to the probability of compromising the HMI times the probability of compromising the RCM. As the probabilities of both should be (much) less than one, their product is a number that is (much much) smaller than either of them.

physical isolation and general physical security of the plant makes it difficult to physically attack the proposed augmented device without detection, while the low speed links limits the amount of visible traffic making it difficult to mount one of the known attacks against the cryptographic techniques.

Even though this solution does not propose any solution to overcome the cryptographic shortcomings, the problems of privacy and computational freedom will be addressed. The legitimate user has full rights and complete access to the secret keys; this is achieved by allowing the user to enter secret keys either using a trusted device or by connecting directly to the device. Both solutions require physical presence at the device, thus making the solution more secure as secrets are configured *before* installation or these secrets are configured by personal who have legitimate access[5] to the devices. Similar solutions can be found in [61, 62], 60].

The next chapter will briefly outline the TCG TPM specification and present several working implementations. This will be followed by a description of the proposed lightweight implementation. Each component of the proposed solution will be reviewed, along with the software needed for the implementation.

[5] Legitimate access here refers to physical access to the PLCs.

7 Integrated TCM for a Network Connected Device

The previous chapter presented a TCM specification by TCG and described some implementations of TCM for PCs and embedded devices. However, these implementations were not designed for resource-constrained applications. This chapter and the subsequent chapters will present a TCM for resource-constrained components. In an attempt to minimize the TCM implementation costs & complexity some modules are removed and some modules' functionality is implemented inside other modules. The advantage of an integrated TPM is that the components could be distributed over different physical locations and still be integrated in the system, helping to provide security while reducing the size (in mm^2) of the implementation – hence reducing the cost. Additionally, placing the components close to or in the system helps reduce the cost of communication over a bus system, avoiding degradation in performance. Similar implementation for wireless sensor networks have been done reported in [59] and also another implementation in [60]. The first of these two implementations is based on a 32-bit architecture and the second lacks the security benefits presented here. A typical implementation is depicted in Figure 5.

Below are fundamental points worth noting when seeking to minimize the TCM:

- The only thing we rely is on is physical security, but given physical security there is no need to implement security in all I/O modules as they are assumed to operate in a secure physical environment. The exception to this is I/O module(s) that are connected to the network.
- There is no need to implement or encrypt the memory contents as the memory is also embedded in the system, thus no one can access it unless they have physical access.
- The TPM components also serve as a crypto engine if the CPU wants to encrypt stored data and backup it on removable media (such as a flash drive). This gives the TPM an additional use other than security within the running system.
- The TPM must be able to communicate and listen to the network in order to perform a secured TPM configuration and update. Drivers and functions should be provided to facilitate such use of the TPM.
- The alias-key generation (AIK) mechanism could be excluded from the system to save on space and make the TPM as lightweight as possible. Since SCADA systems do not need as many identities as typical PCs, the SK could be used on permanent base.
- The Secret Key (SK) could be configured in the system by using a smart card or other trusted device or trusted application running on the machine. This could be used to set up the primary configuration, which precludes direct visibility.

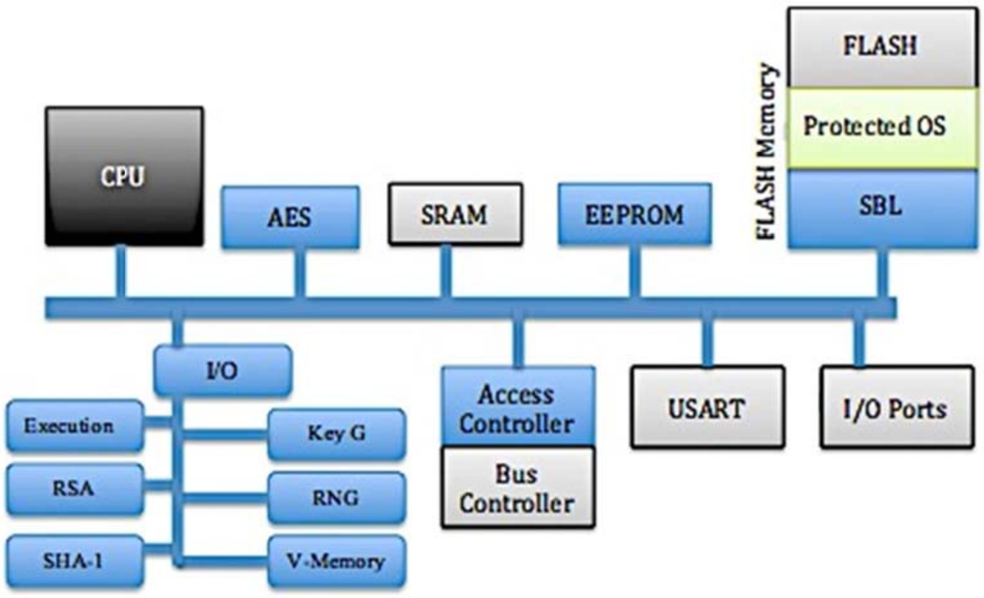


Figure 5: Typical PLC with integrated TCM

7.1 The Access Controller

Moving from the physical security of the network connected devices, which is completely concerned with external security; the next security requirement concerns the internal security of stored data and internal resources. In our proposed model an access controller implements an internal security policy that should protect the memory and resources from unauthorized access. The access control policy decisions should be based on the user's privilege and the current TPM state. The policy should grant or deny access by considering the privilege of the requester, the requested resource, and the mode of the requested resource[32].

The access controller controls all access rights to resources connected to the system bus. Access rights are policies that govern which request is allowed access to a certain part of memory, I/O subsystem, or other component. An access request and interrupt requests are processed on a priority base. Priority is used as a selection mechanism to selected requests in the case of concurrent requests from different components. A typical priority implementation could be first come first served or round robin, and the designer must consider advantages and disadvantages associated with the selected priority mechanism and account for the bus and time penalties associated with the mechanism. A fully functioning access controller must implement both priority and access rights management.

7.1.1 Requirements

As mentioned in section 3.3 page 15, the major components of TPM are the CRT, RTR, and RTS. In this thesis project the RTS is implemented using an access controller to control access to the system memory. The RTS must fulfill the requirements of an RTS as specified in the TCG specification **and** must fulfill the requirements outlined below in order to achieve the desired service. In the TPM implementation the access controller plays a major role in providing security, isolation, and helping to minimize the TPM. These specifications are organized into three requirements, as listed below. This list is followed by a detailed explanation of each of these requirements.

1. The access controller must help isolate the TPM from any unauthorized access.
2. The access controller must help isolate privileged memory for the TPM to utilize exclusively.
3. The access controller must implement the access rights and policies that are specified by the TPM.

7.1.2 Isolation of the TPM Components

The primary requirements of TPM are isolation of system and protection of secrets. The TPM **must** be isolated from the rest of the system. TPM components such as the key generator, private keys, configuration registers, and measurements should be isolated and protected at all times from **any** application running on the processor.

Components such as the encryption engines and public keys need to be accessed by both the CPU and the network module(s). This will allow the TPM to serve as both a security module and provide encryption engines. A network module should take full advantage of the cryptographic engines, so that encryption can be enforced for all external communications without burdening the CPU. As a result the cost of encrypting external communications should be invisible to the execution time of applications running on the CPU.

7.1.3 Isolated TPM Memory

Instead of using a separate memory in the TPM this proposed solution implements the memory using part of the main memory as an isolated memory for the TPM's exclusive use. Working memory for storing configuration register values, storing random number generator state, private keys, and encryption keys must be allocated and the protection policy must be enforced.

Part of the ROM could be used for storing the SK, part of the flash memory could be used to storing other sensitive information such as private keys and RNG random numbers. These locations have to be available all of the time and the protection policy must be implemented to protect these locations from being over written or read by other than the TPM. These design decisions reduce the need for external or additional memory in the TPM to store sensitive information. However, the TPM must itself implement sufficient volatile memory so that all the stored data can be copy to it at runtime[6]. The goal is that the TPM can work without interrupting the CPU plus the reverse operation (copying from volatile memory to primary memory) should take place when there is a change in measurements due to update or new configuration or before any shutdown. Additionally, this design reduces the number of times that the ROM or flash is written, which is a major power savings and prevents component degradation due to multiple write cycles.

[6] The copy operation occurs when the system changes from powered off to power on state.

7.1.5 Implement privilege and Access Control

By implementing a set of rules and policies that specify a memory range, access type, TPM state, and user identity - a set of privilege levels can be configured. These privilege levels govern the access rights to memory and peripheral I/O. The proposed solution implements at least 4-privilege levels. These privilege levels are:

1. **Non-secured privilege**: cannot access any secured section, only allowed to access the non-secured and unprotected sections of memory.
2. **Secure working privilege levels**: no read or write access to the TPM's secure memory; no write access to protected OS memory; can only request access to public key for secured communication. This would be the privilege level the CPU runs in when running a secured OS.
3. **Trusted users levels**: user has established trust to exchange keys and certificates with the TPM. With this privilege the user can change the protected OS, i.e. there is no read or write access to the TPM's secured memory, but the user can read and write protected to the memory containing the OS and the other parts of memory. This would be the privilege level the secured boot loader (SBL) would be running in when updating the protected OS. Further details of the SBL will be given in section 8.1 pages 43.
4. **Trusted machine level**: is the TPM privilege level where access to all secrets and memory locations is granted. The TPM needs to be able to access every memory location in order to perform measurements and check the integrity of stored data (including the OSs and applications).

7.1.6 Implementation

For fast and reliable implementation the access controller must be implemented in hardware and placed in or near the bus controller. The access controller could be implemented on the bus as a separate module or integrated it in the bus controller subsystem. In the latter case the designer would implement the policies in the bus controller. Implementing the controller in the bus controller takes advantage of the closeness to the bus and reduces the cost of communication over the bus. However, implementing the policies in the bus controller could be challenging, as one has to work within the limited configurability of the bus controller. Implementing the access controller as a separate module has an advantage of modularization and openness for versatility, but lacks the advantages of the integrated approach mentioned above. As a result perhaps a hybrid of the two approaches would be ideal, i.e., implementing some of the functionality of the access controller in the bus controller, while implementing some of the functions in a separate module to offer greater flexibility while saving resources.

In either implementation the access controller must take the TPM state, the requester's identity, the requested address range, and access type (read or write) as inputs to its decision. The access controller's output is a granted or denied indicator. The memory ranges could be defined in the ROM and the controller could read these values and enforce the required memory isolation.

7.2 The I/O module

In contrast to the TCG specification where the I/O module provide access control and system isolation, in this implementation the I/O module serves as only isolates the TPM bus from the rest of the system. The I/O module is used as an interface between the TPM components, the CPU, and the shared memory. The I/O module must interface with the bus system in such a way that (controlled) memory access is easily achieved. A request for access to a special TPM component is channeled to the appropriate module. In general the I/O module serves as communications interface between the CPU and TPM. The I/O module structure also defines how the TPM is organized, connected, and commanded. If the TPM is also to be used as a network encryption accelerator, then the I/O module must implement an interface to communicate with the Network module.

7.2.1 Requirements

The I/O module must fulfill the following requirements:

1. Isolate the TPM bus system from the rest of the system.
2. Interface TPM components with the main bus system.
3. Facilitate inter-communication among TPM devices.

7.2.2 Implementation

The I/O module could be implemented as a bus bridge isolating the local TPM bus from the system bus. This bridge would accept the address of the sub module to be accessed, the address of memory operations that will take place; and an op-code instructing the TPM what to do from the main system bus. A driver and API could be provided for higher layer applications to take advantage of the TPM's resources. Since the bus arbiter provides isolation there is no need to enforce isolation policies. However, the I/O module must provide a means to facilitate encryption and decryption to implement encrypted communications over the network, while off-loading these tasks from the CPU.

7.3 The cryptographic engines

The cryptographic engine must provide encryption and decryption, key generation, data signing, and implement hashing. Data-sealing and memory management could also be implemented. As the working memory of the TPM is limited the available working RAM must be managed and monitored. The engines can be made to serve both the TPM and the CPU with priority given to the TPM. The TPM performs data sealing and unsealing at boot time and after any update with the help of a SBL and the TPM execution engine. The components of the crypto engine are:

- RSA engine,
- SHA-1 engine,
- RNG engine, and
- Key Generator (optional).

The Secured Hash Algorithm (SHA) was developed by U. S. National Institute of Standard and Technology (NIST) in 1995 published in [63]. It remains the most widely used hashing function [64] throughout the world. NIST published SHA-0 in 1993, but it has been deprecated because it was found to have serious security flaws. Even SHA-1 has shortcomings that were discovered in recent years. For example, work by [55] argues that with enough resources the hash could be broken in a couple of weeks. To mitigate these shortcomings NIST has published SHA-2 (also known as SHA-256 or SHA-512) that generates a larger digest output.

The SHA-1 algorithm work by first adding a bit '1' and '0' to the message to make the total size similar to 448 mod 512. Next the messaged is chopped into chunks of 512 bits, which are then grouped into 16 groups of 32 bit words. The 16 groups of 32 bits are converted into 80 bits by applying some logical operations, such as Exclusive OR and left rotation. Then the 80 groups of 32 bits are hashed and appended to either the initial vector or a previous hash output resulting in a 160-bit hash of the message. It has been estimated that it would require 1.4 x 10^{48} guesses to recover the data from the hash through a brute-force attack and 1.4 x 10^{24} guesses with birthday attack[65].

SHA-1 was chosen over other hashing functions by TCG because of three main reasons. First TCG specifies the need for SHA-1 hashing for interoperability[31]. Second the long usage of SHA-1 has shown the reliability and superiority of SHA-1 over other hashing algorithm such as MD5[65]. Third SHA-1's modular implementation makes it easy to upgrade, modify, and replace it with no or little changes to the rest of the system.

The RSA module must implement encryption and decryption of data using the private public key (asymmetric) encryption. Although the TCG specification[31] recommend keys with a length of 2048, the specification also supports smaller keys, which is advantageous for reducing the RSA size. The specification and the detail of the implementation should be left to designer as should the selection of key size these is to allow the designer to take advantage of the tradeoff on the performance speed and implementation area size. Since RSA works using the difficulty of factoring large number, the larger the key size the more difficult it is to guess the key.

The encryption technique in the RSA encryption algorithm is based on the difficulty of factoring a large number. The public key is *puk(n, e)*, where *'n'* is a product of two distinct prime number *q* and *p* and *'e'* is any number between 1 and *(p-1)(q-1)* where the greatest common factor for *e* and the upper bound ($\phi(n)=(p-1)(q-1)$) is 1. And the private key is *prk(n, d)*, where *'d'* is the *modular multiplicative inverse of e (mod* $\phi(n)$*)*. Encryption on message *'m'* is done $c= m^e \ (mod\ n)$ and decryption is done $m= c^d \ (mod\ n)$. So the RSA engine must be able to do both encryption and decryption by loading the appropriate keys required.

Although analysis for wireless sensor network done by [56] argue other cryptographic solutions such as elliptic curve digital signature algorithm provide better protection than RSA for equivalent key size. Performance comparison done in [66] argue though elliptic curve and Diffie-Hellmen provide smaller area implementation than RSA, RSA provide faster certificate authentication for comparable security.

The RNG must provide random values for nonces, key generation, and randomness in signatures. The specification and requirement for providing entropy and maintaining randomness are specified in detail in the TCG specification. The nonce could be used to XOR the data before or after encryption and to ensure that data is not encrypted several times with the same key. This is useful to mitigate replay attacks that could be attempted by capturing and retransmitting previously sent packets. The other output of the RNG is to provide the key generator with a random number to generate new keys to be used as the AIK.

The key generator is responsible for generating the AIK key, which will be used as an alias to the primary key or SK. The key generator generates a private-public pair of keys and the private part is encrypted with SK and stored in memory, while the public part is sent to the CA for certification and authentication. In the TCG implementation the TPM could register multiple AIKs and use them for different communications. However, to make the TPM more lightweight one can remove (these solution propose removing) the key generator and use the nonce as an alias by ciphering data using either XOR or one time AES keys generated by the RNG. Data could be encrypted or ciphered by XORing with a nonce making the system more secure. The nonce could be stored in the RCM with the PLC's public key and other parameters. This implementation takes less memory space than storing a complete AIK, thus making it ideal for a memory-constrained implementation. The proposed solution removes the AIK capabilities and uses nonce instead. This design decision saves a lot of resources.

7.3.1 Requirements

The requirements of the cryptographic engine and its subcomponents are:

1. The crypto engine generates measurements to be used as CRT.
2. The SHA-1 module must provide hashing for measurements and integrity tests. The individual requirements are:
    - Hashing the stored secrets keys and private keys.
    - Hashing of all stored measurements to prove integrity of measurements.
    - Hashing for different memory sections, such as SBL and secured OS, to generate measurement.
    - Hashing of secured update to verify integrity.
3. The RAS module must encrypt data using the private key or using an alias key generated in the TPM.
4. The RNG must generate random numbers to be used in the signing and alias key generation.

7.3.2 Implementation

There exist several hardware based lightweight and fast implementations of the SHA-1 algorithm, RAS, and RNG; therefore integrating any compatible version would be a choice for the designer. The designer should choose and limit the number of keys needed for encryption and decryption based on the nature of the network that is implemented on the system. The choice of key length is always a tradeoff against area, power consumption, computational burden, and data security. As mentioned in the future work section exploring other cryptographic techniques and evaluating the performance and advantage each offers would help to further reduce the TPM size, while maintaining the desired security level.

7.4 The Execution Engine

The execution engine supports all the operations taking place on the TPM. The execution engine also runs in parallel with the CPU to facilitate secured communications. Since the TPM is implemented as a master in the system (with respect to the system bus), the execution engine can initiate memory read and write with the rights and privileges governed by the Access Controller. Although not part of this thesis project, the execution engine could be configured as an encryption engine to encrypt and decrypt communications over the network. With direct access to the network module of the system more advanced key management schemes and system updates could be possible.

7.4.1 Requirements

The requirements of the execution engine in this system are:

1. Implement a means to measure the reliability of the TPM,
2. Implement a means to measure the secret keys and SBL,
3. Implement a means to protect unsecured boot, and
4. Implement an interface for higher-level applications to establish secured updates and communication with other connected devices.

7.4.2 Implementation

The execution engine should be implemented as a set of op-codes to be interpreted with the help of an address generator and a program counter. The pseudo code of the execution engine is given below.

```
Boot
Measure execution op-code
Compare with previous measurement
If okay
    Set boot flag bit0 to high
    Measure the secret key
    Compare with measurement
    If okay
        Measure the SBL
        Compare with previous measurement
        If okay
            Set boot flag bit1 to high
            Go to Execute
        else
            Set boot flag bit1 to low
            Go to Exit
    else
        Set boot flag bit1 to low
        Go to Update
else
    Set boot flag bit0 to low
    Go to Exit
Endboot

Update
If physical presence bit set high
    Measure secret key
    Store measurement
Go to Exit
Endupdate

Execute
Listen for a request via the I/O module
Respond to request
```

```
        /* request      could be measure_data,
                                 encrypt_data,
                                 decrypt_data,
                                generate_AIK,
                                generate_nonce, ...
        that use the services provided by the TPM*/
loop back to listen for another request.
endexecute

Exit
Save RNG random state.
Clear RAM
Exit
endExit
```

8 Software components

In the previous chapter, the hardware solutions proposed was presented in detail. The necessary requirements and implementation criterions were outlined. This chapter will address the secured boot loader (SBL) that will be loaded on the main processor and the reliable computing machine (RCM) that will monitor the overall performance of the system. And also the implementation detail of the secured protocol (SP), the software that is run by the TCM execution engine will be presented.

8.1 The Secured Boot Loader (SBL)

The SBL is a measured application with known trust and integrity. It might never be updated or updateable only with known trustworthy version. In order to ensure the later, each current version must be able to check the signature of the next version. The SBL establishes the secured I/O and secured memory that will be used as the security based for all secured applications. The SBL initiates the TPM measurements of the TPM's secured memory locations, including the SBL and secured OS. The boot loader will use these measurements to make other decisions. The SBL will also communicate via the network to update a secured OS. After a secured boot the SBL will pass ownership of the system to the secured OS and the secured OS will be responsible for the remaining secured communications. As noted previously ideally the network module could be configured to encrypt and decrypt all communication without involving the CPU.

8.1.1 Requirement

The requirements of the SBL are:

1. Become the RTR by reporting discrepancies in measurements stored in the RTS,
2. Establish the root of trust for higher level application such as OSs by assuring secret and measurement integrity,
3. Establish trust and run secured communication with other machines in order to securely update the OS,
4. Securely update RTS and CRT measurements, and
5. Securely boot the OS.

8.1.2 Implementation

In addition to the standard boot loader algorithm the implementation of the SBL should incorporate the following algorithm. This algorithm is summarized by its operations in the following scenarios:

1. **TPM initiate**
   Initiate the TPM by asking if it is ready. The TPM runs integrity measurements on all stored secrets, measurements, and data stored in the isolated memory. If all of the secrets are intact, this means that the CRT is established and then the TPM measures the SBL. If the results of this measurement concur with the stored measurement, then the TPM responds to the boot loader's requests.

2. **Check and Reporting**
   The SBL requests the integrity of secrets and measurements and generates an error report (if necessary). The SBL requests the integrity of TPM, SBL, and OS and then generates an error report before exiting (if necessary). This would establish the RTR would report errors occurring in the TPM.

3. **Secured updating**
   After making sure that the SBL and all secrets are intact, the SBL checks via the network interface for a possible update request. The SBL will continue to listen for some period of time before either jumping to the protected OS or exiting. If an update request is received the request is processed and the updater is verified through a secured protocol. The secured protocol will communicate with the RCM and verify the updater and the update signature. After the update is received is measured, then the protected OS is updated. After this the TPM updates its measurements with a new measurement of the OS and the SBL jumps to the start of the protected OS. If at any point during the update there is a disagreement between measurements, then the update is aborted and the SBL reports an error and exits. Once the OS is started the responsibility for running secured application and secured communication will be the responsibility of the OS.

A flowchart showing the SBL's operation is shown in Figure 6.

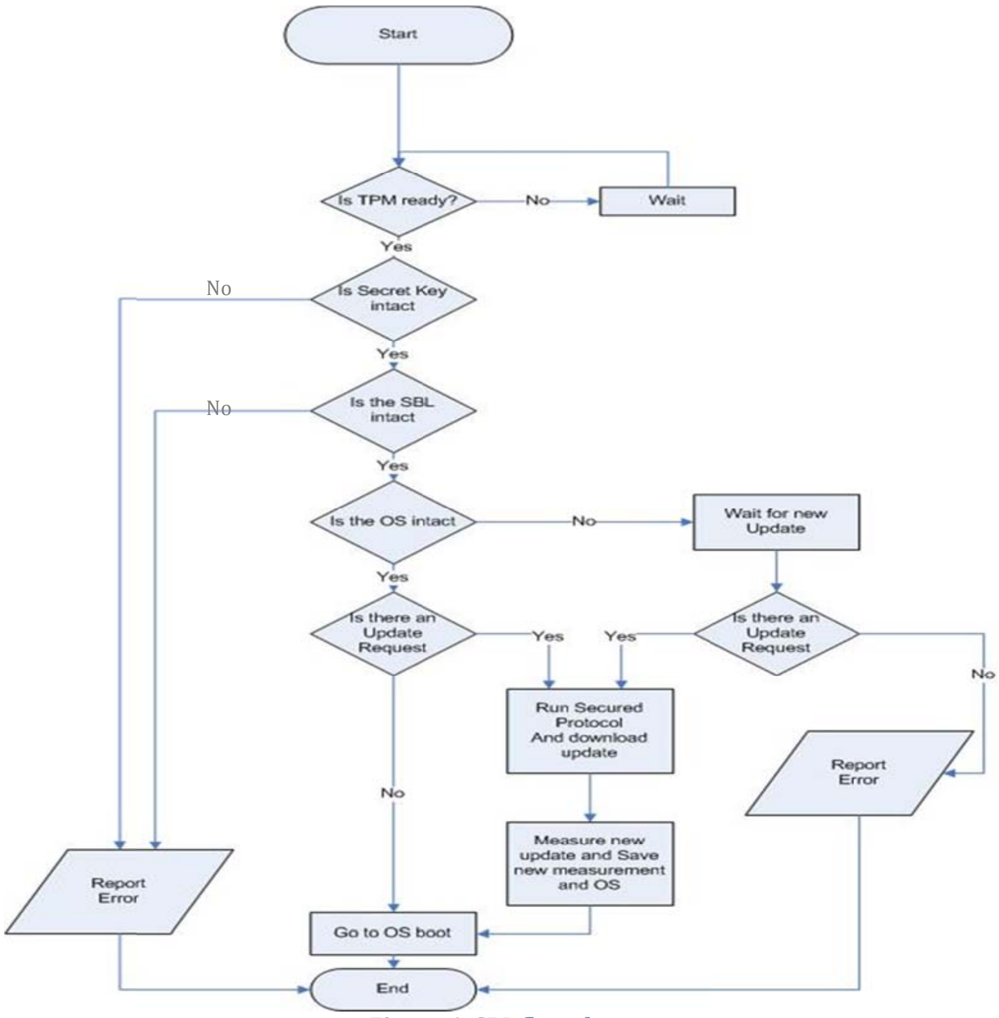


Figure 6: SBL flow chart

8.2 The Secured Protocol (SP)

The SP is a set of requirements and procedures followed by the trusted machine for secure exchange of updates, certificates, and keys. The SP is proposed with the intention of minimizing the number of keys and certificates that need to be stored in the PLCs in order provide secured communication. The SP usage diagram and message exchange is summarized in Figure 7. In a typical scenario the Updater will communicate with the RCM, establish trust, and acquire PLC public key and nonce. When trying to update the PLC the updater will send the signature of the update to the PLC and the un-compiled code to the RCM to compile the code and generate a signature. After this the PLC will forward the update signature to the RCM to verify the update. If this signature is verified, then updater is allowed to update the PLC and the update is sent to PLC in encrypted form with the public key and nonce offered to the updater.

The implementation of the SP could take advantage of the fact that the authentication and key distribution are handled by the RCM in two ways. The first would be to run different encryptions such as RAS 512 on the PLC and RAS 2048 between the HMI and RCM. This would enable the TPM module to be even smaller and reduce the processing burden on the TPM. The second is the PLC does not have to store any certificate or key other than that of the RCM, thus relieving the problem of constrained memory.

The format of the messages exchanged could be:

Request for update PLC message

Encryption with PLC public key begin

Updater ID (MAC)
Receiver ID
Nonce unique random update number
Update SHA-1 hash
Message hash

Encryption end.

Request to verify updater message to RCM

Encryption with RCM public key begin

Updater ID (MAC)
Sener ID
Nonce unique random update number
Update SHA-1 hash
Message hash

Encryption end.

Response Updater verified

Encryption with PLC public key begin

Updater ID (MAC)
Updater public key
Update validity (update valid or not)
Nonce unique random number
Update SHA-1 hash by RCM
Message hash

Encryption end.

```
Response updating Granted to HMI
    Encryption with HMI public key begin
        PLC ID
        Nonce unique random number
        Message hash
    Encryption end.
Request to register nonce or AIK
    Encryption with RCM public key begin
        PLC ID
        Nonce unique random number OR AIK
        Valid until number
        Message hash
    Encryption end.
Response to request to register nonce or AIK
    Encryption with PLC public key begin
        Request accepted or rejected
        Message hash
    Encryption end.
```

8.2.1 TPM as a Cryptographic Engine

As described earlier, the TPM module could also serve as a cryptographic engine facilitating encrypted communication between components. The resources such as the RSA and SHA engines could be available for the users to take full advantage of these components. The opportunities presented to the user through the TPM cryptographic engine range from secured reporting to encrypted backup and secured network communications[7]. This functionality would help the TPM achieve the desired features implemented in "position embedded " as mentioned in [67], [68]. Although these implementations are very attractive, they suffer from a loss of integrity and having an extra module in the communication path would further degrade the communication speed. However, using the TPM side by side with the CPU to provide the "position embedded" features profit by accessing data before transmission is started, which could save a lot of time.

[7] The secured communication can be achieved either by using the TPM as a full time encryption engine, which could result in slow communication. Or using the TPM to exchange a symmetric key, which afterward could be used to continuously exchange data.

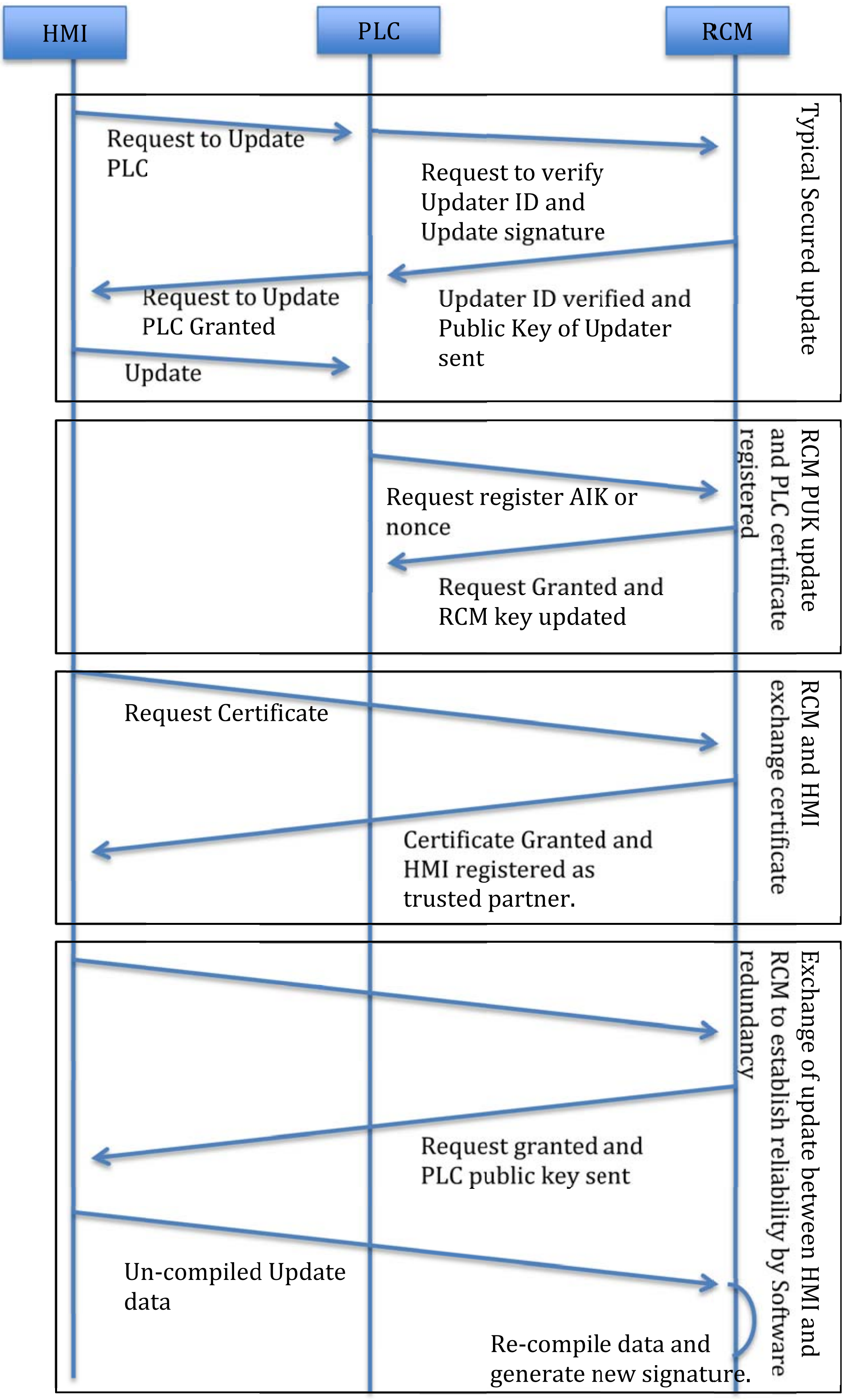

HMI
PLC
RCM
Request to Update PLC
Request to verify Updater ID and Update signature
Updater ID verified and Public Key of Updater sent
Request to Update PLC Granted
Update
Typical Secured update
Request register AIK or nonce
Request Granted and RCM key updated
RCM PUK update and PLC certificate registered
Request Certificate
Certificate Granted and HMI registered as trusted partner.
RCM and HMI exchange certificate
Request granted and PLC public key sent
Un-compiled Update data
Re-compile data and generate new signature.
Exchange of update between HMI and RCM to establish reliability by Software redundancy

Figure 7: Flow graph for secured update

8.3 Reliable Computing Machine (RCM)

The reliable computing machine lies at the center of the proposed security solution. The RCM implements the core security mechanisms proposed by this solution, specifically: updater attestation and software redundancy. The RCM is to be executed on a more robust machine with greater capabilities and more advanced communication with the HMI and updaters. The RCM can attest to the trustworthiness of the updater. Additionally, by using a redundant software module the RCM could verify code integrity, but making any code breach more difficult.

The RCM communicates with the PLCs and the HMI. The RCM could use different security polices for communicating with the PLCs than when communicating with the HMI.As a result one could implement a less complex security for the communication between PLC and RCM and more complex security policy between the HMI and RCM. Sharing the security polices between the RCM and the PLC allows for simpler security policy implementation in the PLC. This is a greatly desired as the PLCs are constrained in all aspects, including memory and computational power. Any updater has to be verified by the RCM, which can utilize a more complex security policy, while relieving the computation burden of the PLCs.

The RCM executes a redundant software module, which attests to the update code's integrity by recompiling the update from source code. The RCM also maintains a database of update code and generates a report on the trends and changes in the updates. The database can compare the previous and current update and notify users of the difference and changes. These reports could be used as additional system monitoring mechanisms, while the primary security based upon recompiling the update and generating a signature is used to verify that the signature sent to the PLCs matches the authoritative version received by the RCM.

We achieve the security goals and requirements that we desire by combining the solutions implemented on the RCM and PLCs, i.e. exploiting software redundancy, the update trend database, and TPM. By using encryption together with a nonce the proposed solution tries to mitigate the vulnerabilities of SCADA networks. By implementing user authentication and software redundancy the proposed solution tries to mitigate the vulnerabilities imposed by the control software and underlying OS. With the help of the database and update history the proposed solution tries to monitor and counter check the system's conformance to its expected changes in operation. The next chapter will analyze the security protection that can be achieved and evaluate the implementation cost and feasibility of the proposed solution.

9 Analysis and Implementation

This chapter analyzes the security protections offered by the proposed system in comparison with the vulnerabilities presented in previous chapters. Although most of the security analysis is informal, it should serve as a good starting point to design a more formal analysis and it should provide a sufficient base to analyze the strengths and weakness of the system. The sections following the security analysis describe the implementation of a simple prototype of the proposed system and measure the cost of this implementation with respect to system resources, system performances, power, and area.

Comparing this simple implementation against the NGSCB and TrustZone, the proposed solution has fewer components and these components are more tightly integrated into the system. Although the solution strongly adheres to the specification and requirements outlined by TCG, there are some deviations regarding key size, memory arrangement, the functions of some modules, and key utilization. However, the proposed solution shares the techniques of the TCG specification common to NGSCB and TustZone. These shared techniques include CRT, RTR, and RTS, which are implemented using the strategies presented in the previous chapters.

9.1 Security Analysis

Although a formal security analysis has not been made of the proposed solution an informal analysis is given below. A formal analysis would include defining attack vectors for the various types of security breaches mentioned earlier in the thesis. A complete formal security analysis is left as future work. Our informal analysis indicates that the proposed solution offers the following:

Suitable polices to control and manage secrets: as the secrets keys are entered into the PLC only once and are never accessible by any means[8], managing and keeping the secrets intact is relatively easy. In addition, the keys could be configured using another key generating device such as smart card or key generator making key configuration even more secure.

Adequately designed control system: the proposed PLCs implement adequate cryptographic capability to encrypt and decrypt all external communication. Since the security module is already embedded in the system there will be no need for an additional module. The additional memory control imposed by the Access Controller provides greater security and internal data integrity than a system without such an access controller.

Secured remote access: remote access is more secure than current PLCs as anyone trying to access the PLC has to verify their identity and use encrypted communications. This makes remote communication more secure and reliable because once the remote host is authenticated other protocols can be used to make communication reliable. Additionally, the use of an encrypted hash and nonce makes the data communicated more reliable. An attacker would have to establish trust with the RCM, which is much more difficult than the case when attacking existing solutions where the attacker only has to authenticate itself using password and dialup security.

[8] As per section 6.2 we assume that there is sufficient physical security to prevent physical attacks on the device itself.

Adequate control of the software used in the control system: although not formally tested, the software redundancy seems to provide adequate control of the software used in the control system. Any successful attacks would now have to compromise two different machines running in different environments. In addition, the attack has to lead to an identical signed hash in order to be accepted by the redundant software security.

Mechanism to detect anomalies: there are two means of detecting anomalies: using the database history of updates and code signing. The database utilizes a trend-detecting algorithm and any variation outside of prior trends will generate a message to a technician. Code signing is used to detect any changes in code while in transit.

Data and update authentication: all communication is authenticated using cryptographic techniques and the RCM. Reliability of every update is authenticated using a signature, nonce, and encryptions. This should make a malicious update infeasible. Additionally, there is a restriction of unauthorized reading of memory that when combined with encrypted communications makes it impossible for any malicious application to gather information about the PLC.

The fact that keys are isolated, the OS is protected, and updates are encrypted and verified gives the system better security that systems without these features. These features guarantee that the proposed solution fulfills the requirements and recommendations outlined by AGA and TCG [69], [31].

9.2 Implementation

Prototyping a fully functional PLC with all the redundant features and modules is difficult for several reasons. Most PLCs are built based upon specification and modules that are only accessible to a specific company. Additionally, most designs incorporate modules that are protected by intellectual property rights owned by their respective companies. Furthermore, PLCs designed for harsh industrial environments implement several redundant modules to minimize fault and downtime. So implementing all of these modules simply for laboratory testing would require a great deal of resources. For these reasons the implementation and testing were done by simulating a simple soft-core processor with limited I/O and memory.

This approach to implementation has its own advantages and disadvantages. The advantages include simplicity of implementation, versatility of operating speed and frequency, flexibility of bus width and word size, etc. The major disadvantage is that the soft-core CPU is not as optimized as the CPUs commonly found in PLCs, thus the size of the soft-core CPU might be more or less than an actual CPU in a commercial PLC. The operating frequency of simulation could be much faster than the typical PLC's operating frequency; hence the timing evaluation needs to be taken with a grain of salt. In addition to the soft-core CPU itself, the proposed solution was implemented with open source components, which are designed for general-purpose applications and small (home) projects. For this reason most components are only a simple implementation of the algorithms and solutions proposed, hence these components have not been optimized for size or power. However, as stated above this implementation serves as ideal ground to test *basic* functionality and to evaluate the proposed solution's overall performance.

The prototype was developed using two proprietary software packages, namely ModelSim® and Quartus II®. Both programs were accessed using the student university license found on the server license.imit.kth.se. Since these licenses are for limited functionality there were problem compiling files when using larger FPGA libraries. ModelSim® is a simulation program from Mentro Graphics®, it was extensively used to develop simulation and measure execution time. Quartus II® is a synthesis and analysis tool for hardware designs from Altera®, it was used to convert the VHDL code in to register transfer level. It was also used to generate the circuit layout of the prototype and estimate the size[9] of the components. Additionally, two sub-components of Quartus II®, the SOPC builder and MegaWizard, were used to develop standard components such as soft core CPU, RAM, ROM, Flash interfaces, and I/O components.

For the implementation and evaluation a 32 bit soft core CPU called NIOS II/e from Altera was used as the core processor. This processor has no hardware multiplier nor branch prediction mechanism and most of its implemented features are rather basic; additional details can be found in [70]. 4KB of RAM, 4KB ROM, and 512KB of flash memory were incorporated. To implement the TPM open source cores from Opencores.org were used. The TPM was implemented using open source SHA, RSA, and RNG components found in the Opencores.org library. A 512 key RSA engine was used as a compromise, which has the advantage of reduced execution time and area at the cost of somewhat weaker security. The Access Controller and Execution Engine were prototyped and simulated. Although the RCM was not implemented, sufficient simulations and timing measurements were done on the prototyped system to understand the system's overall performance. In the next section we will examine and evaluate the size and response time of the implemented system.

9.3 Results and Measurements

For sake of comparison the proposed solution's performance was compared with the performance and specification of Siemen's SIMATIC S7-300 gathered from [71]. The SIMATIC S7-300 host features a CPU with a clock speed ranging from 30 MHz to 750 MHz; memory capacity ranging from 42 KB to 2.6 MB; and communication support for PROFINET and Ethernet. Typical communication speed is 12 Mbps via PROFIBUS interface and 19.2 Kbps single duplex (38.4 Kbps double duplex) Point-to-Point serial line communication. The instruction processing time ranges, depending on the type of process (bit operation to floating point operation), from 0.1 µs to 1.1 µs on the slowest CPU to 0.004 µs to 0.04 µs on the fast CPU.

The implemented prototype is estimated to use 15K logic elements, 10K dedicated registers, and 129K memory bits when implemented in an Altera FPGA. Of these resources the CPU takes 22.58% of the total logic elements, 18.30% of the total dedicated registers, and 85.3% of the total memory bits. The area of the TPM is large when compared with the area of the CPU. However, it should be emphasized that the implemented CPU lacks most of the critical features found in PLC (such as analog to digital converters, analog and digital I/Os, network communication interfaces, several timers, and multiple counters). Additionally the CPU that has been used has no replication or voting to support higher reliability. For all these reasons the area requirements of the TPM might in practice be insignificant, if all these features were included in a commercial CPU. Most of the area is taken by the RSA engine, which

[9] Size meaning areal size, which is measured in terms of registers, logic elements and memory bits.

takes 58.4% of the total logic elements, 81.1% of the total dedicated registers, and 10.3% of the total memory bits. These numbers show there is a room for farther reduction of the size of the TPM by improving and optimizing the RSA engine.

Table 2: TCM area and execution time

Component	Area			Execution time in cycles
	Logic element	Dedicated register	Memory bit	
RSA engine	8 895	7 362	7 744	100K
SHA-1 engine	1 309	984	0	88
RNG	512	512	0	1
I/O module	<100	<100	0	1
Execution engine	481	181	10 240	1
Volatile-RAM	0	0	34 304	1
Access Controller	<100	<100	0	1
CPU	3 325	1 839	77 312	1

Comparing the execution time on a SIMATIC S7-300 CPU, the fastest communication interface, which is the PROFINET interface, is typically 12 Mbps and the slowest communication interface, which is the Point-to-Point interface, is 38Kbps at full duplex [71]. On the fastest link speed it would take 42.6 µs (512 bits/12*10^6 bit/sec) to send 512 bits of data while it takes 13.47 ms (512bit/38*1000bit/sec) on the slow link. The fastest operation (which is a bit wise operation) on the slowest SIMATIC S7-300[10] CPU takes 0.1 µs (corresponding to simulation clock rate of 30 MHz) while on the fastest CPU it only takes 0.004 µs (corresponding to simulation clock rate of 750MHz). In contrast the slow operations (such as a floating point operation) take 1.1 µs and 0.04 µs on the slowest and fastest SIMATIC S7-300 CPU (respectively). In the implemented prototype it takes 88 clock cycles to fully load 512 bits of data to the SHA and to retrieve a 160 bit hash data. In contrast it takes about 100K clock cycles to perform a RSA encryption on 512 bits of data and to retrieve the resulting 512 bits of encrypted data.

Table 3: Siemens s7-300 CPU family execution time

Process execution	CPU 312	CPU 315-2 DP	CPU 319-3
Bit operation	0.1µs	0.05µs	0.004µs
Word operation	0.24µs	0.09µs	0.01µs
Fixe-point operation	0.32µs	0.12µs	0.01µs
Floating point operation	1.1µs	0.45µs	0.04µs

[10] All the data used here or used for the table can be found on the manufacturer website and also on [71].

Using the clock rate as a limiting bound, the effect of adding the TPM on the typical operation of the PLC is evaluated as follow. As a typical scenario, 512 bits of data after a bit-operation are hashed and sent through the PROFINET interface. Since PROFINET is a serial interface, while the rest of the system performs operations on words, producing outputs faster than the communication speed should guarantee smooth operations (i.e., that the required processing would not constitute a bottleneck). A bit-operation at the slow clock rate of 30 MHz can produce 512bits in 1.6 µs, which would equivalent to serial communication at 320 Mb/s. In comparison, the slow floating point operations at a clock rate of 30 MHz would take 17.6 µs to produce 512 bits corresponding to a serial communication rate of 29.09 Mb/s. The SHA-1 engine takes 2.931 µs (88 cycle at 30MHz) to produce a 160 bit-hashed value, corresponding to a serial communication rate of 229.27 Mb/s. As the PROFINET connection is running at 12 Mb/s it takes 56 µs to send the 512 bits of data. Since both operations (bit or floating point operation and SHA-1 operation) are faster than the communication link speed adding SHA-1 hashing will not introduce any significant delay even for the slowest clock rate CPU.

Considering the same scenario using the RSA engine rather than the SHA-1 engine, we find that encrypting 512 bits of data takes 3.33 ms and 133 µs on the slowest and fastest SIMATIC S7-300 CPU (respectively). This speed is very slow in comparison to the typical PROFINET communication. Even if a low speed of 3.84 Mb/s could be negotiated for the fastest CPU the rate for the slowest CPU 153.75 Kb/s is unacceptable for PROFINET communication. However, this data rate is fairly fast for point-to-point, asynchronous, or dial-up modem, whose speed is typically only a few Kbps (typical dial-up modem speed being 56Kb/s and point-to-point speed is 38Kb/s for full duplex). One should also take in to consideration the fact that the RSA engine is using a 512 bit key encryption so using smaller keys would farther enhance the execution time, while using longer keys would farther degrade the performance.

Table 4: Summary of execution performance rate

Execution scenario	**CPU 312**	**CPU 319-3**
CPU Bit operation ≥ SHA-1 hash ≥ PROFINET network.	Not affected 12Mbps	Not affected 12Mbps
CPU Bit operation ≥ SHA-1 hash ≥ Point-to-Point network.	Not affected 38Kbps	Not affected 38Kbps
CPU floating point operation ≥ SHA-1 hash ≥ PROFINET network.	Not affected 12Mbps	Not affected 12Mbps
CPU floating point operation ≥ SHA-1 hash ≥ Point-to-Point network.	Not affected 38Kbps	Not affected 38Kbps
CPU Bit operation ≥ RSA Encryption ≥ PROFINET network.	Affected 153.75Kbps	Affected 3.84Mbps
CPU Bit operation ≥ RSA Encryption ≥ Point-to-Point network.	Not affected 38Kbps	Not affected 38Kbps
CPU floating point operation ≥ RSA Encryption ≥ PROFINET network.	Affected 153.75Kbps	Affected 3.84Mbps
CPU floating point operation ≥ RSA Encryption ≥ PROFINET network.	Not affected 38Kbps	Not affected 38Kbps

It would be interesting to evaluate the time taken by the RSA engine in the same scenario, but in the reverse direction – i.e., when data is being received instead of being sent. Now the slow communication speed will be the major factor in the timing, no matter how fast all the components run they cannot run faster than the input rate (i.e., the communication link speed). Even with the fast communication speed of 12Mb/s of PROFINET it takes 2.667 µs to deliver a word to the RSA engine. This would limit the operating speed of the RSA to 0.375 MHZ (1/2.667 MHz). The RSA engine takes 100K cycles to deliver a 512-bit cipher (in this case decrypted data) taking a total time of 0.2667 s corresponds to a serial link data rate of 2 Kb/s. Unfortunately, this is a very low data rate and the performance is even worse if a slower link data rate is used.

Applying the same calculations for SHA-1 and PROFINET running at 12 Mb/s, the operation rate of the SHA-1 engine will be 2.182 Mb/s. Although this is also a very slow rate compared with the slowest operation (the slow floating point operation taking 1.1 µs) of the CPU, the SHA-1 performance does not degrade the system's performance. Unlike RSA a decryption hash calculation could be run in parallel with other operations. However, if the immediate result of the SHA-1 is requested the response time might be as large as 240 µs.

The above are preliminary results. In an actual implementation by using pipelining and hardware optimization the size of the TPM could be farther reduced and the performance enhanced. Although these results show that the SHA-1 engine does **not** cause any constraints on the communication performance, the RSA engine severely degrades the PROFINET communication but it dos not affect slower communication links. As shown in this thesis the security benefits and flexibility outweigh the additional cost of the increased area and lower performance. Once a reliable trustworthy boot and update mechanism is standardized other faster encryption and security mechanisms can be used. This would result in somewhat slower boot and reprogramming times, but allow rapid communication and more secure execution.

One possible mechanism to tackle the latency is to run the RSA engine with faster speed than the CPU. In this scenario the speed of the RSA engine could be several times faster than either the CPU or the communication interface making the RSA not the bottleneck. Even though it is impossible for the RSA engine to take data faster than provided by the slower modules (now the CPU and the communication interface), but the computation speed will be highly enhanced. For instance the RSA running at 1GHz would only need about 100µs to complete a 512bit RSA operation requiring 100k cycles. This is assuming that all the data is available at the given speed. Plus all outputs are collected at the same speed.

9.4 Discussion

As it can be seen in section 9.3 the area cost of the cryptographic engine is several times more than the cost of the CPU and these costs are farther enhanced when several redundant modules are used. Most PLCs must implement several versions of the modules and components to provide reliability and fault tolerance. As the cost of the TCM so high the actual cost would be several times this as multiple versions of it are implemented. However, I would like to argue that the security benefits and the performance speed of the proposed implementation outweigh the shortcomings. This is affirmed by the two alternative solutions proposed below.

One alternative solution is to replace the RSA engine with a small core processor. Using this small core the RSA algorithm could be implemented in software. This solution has the primary advantage of reduced size, which was the major drawback in the proposed system. But it is argued in [72] that though the area cost of implementation for RSA is high, the cost in execution time and power is even higher when implemented in software. The latency of the software implementation of the RSA stated in [72] is very high and when combined with the slow PLC communication the performance of the system would be unacceptably slow. There could be a middle ground where both cost of area and latency are tradeoff. Doing part of the computation in software and part of it in hardware could do this, but this will be left for future work.

A second alternative is using a RSA server, which is used to do the RSA computation. The PLC and the server can communicate with each other using a shared symmetric key. The PLC can either send its private key and the message it want to be encrypted/decrypted or all the private keys will be stored on the server and the PLC only sends a message for encryption/decryption (a similar implementation is given in [73]). This is a far better solution than the above because it is saves both time and implementation area. The shortcomings include the fact that the PLCs will be dependent on and tied to the RSA server. Also by placing the private keys on the RSA server this server might be a valuable target and hence farther exposer the system attacks. If an attacker manages to gain access to this server it would gain access to not one but potentially all the PLCs private keys making the server a clear target for attacks. This solution lacks the benefit of the TPM, which stores the key in hardware and never exposes or shares it with anyone.

10 Conclusions and Future work

10.1 Conclusions

Regardless of the many security solutions and polices implementations the need to protect SCADA system remains a large task. The vulnerabilities in SCADA systems and the fact that SCADA systems control very critical systems make SCADA systems an ideal target for attack. The vulnerabilities, as mentioned in chapter 1 and chapter 5, emanate from two sources, namely the network communication used in the SCADA network and the OS that is used to monitor and control the SCADA system.

The fact that the network system typically uses simple text to communicate, the fact that the current SCADA systems depend on password protection, and the fact that the SCADA network is frequently connected to the corporate network all increase the vulnerability of the SCADA system. The integration of a gateway device that connects the SCADA network to the Internet breaks the isolation on which SCADA system security relied. The propriety network protocols running on low speed serial links made it hard to implement protection that is more advanced than simple passwords. Additionally, the exchange of messages using standard messages makes it easy for anyone to eavesdrop on the network communication if they can get physical access to the network.

Exacerbating these problems the underlying OS and management software also added to the system's vulnerability. Even with complex security policies and advanced antivirus software protection, malicious worms and Trojan horses have managed to gain access to and managed to compromise SCADA systems. In addition to viruses, cyber attackers have been launching deliberate attacks to exploit the vulnerabilities of the OS. These attacks have managed to cripple SCADA networks forcing the termination of normal operations. Attackers have not only exploited the vulnerabilities via Internet, but also they have compromised SCADA systems using compromised removable media devices, such as USB sticks containing flash memory.

Mitigating these vulnerabilities within the limitations of the system has been the abstract goal of this work. To achieve this goal, two different solutions were proposed. The first is integrating TCM into the PLCs and utilizing a RCM that implement software redundancy to reliably verify the trustworthiness of each update. With the features of the TCM most of the vulnerabilities due to the network communication were mitigated to a very large extent. Mitigation techniques include data encryption, memory protection, secret and key isolation, and measurements of the data to attest to its integrity. The software redundancy mitigates to some extent the ability of an attacker to compromise the PLC as the attacker must compromise both the updater and RCM to be successful. This software redundancy offers a good error detector; thus although the attacker might be able to compromise the PLCs or the RCM - the updater will prevent the system from updating the PLC unless both are compromised such that they yield the same answer. Compromising only one of the two will lead to error, which the human operators can address. However, ensuring that the human operators actually address all of the flagged errors is crucial to securing the SCADA system. Overall the proposed solution provides very good security, while mitigating most of the vulnerabilities of existing SCADA systems.

Especially with the limited resources and constraints imposed by PLCs, when it comes to designing a secure system one should understand that implementing a foolproof system with a single module is very challenging. However, by focusing on the most significant vulnerabilities we believe that the proposed solution mitigates the majority of attacks and improves the overall system security of a SCADA system. In the proposed solution the combination of the RCM and TCM can also mitigate some of the major vulnerabilities due to network communication. However, this is not sufficient in itself to fully protect the PLCs, there must also be a suitable gateway that prevents unauthorized and inappropriate communication between nodes outside the SCADA network and nodes inside the SCADA network.

Although in the previous chapters, there were many suggestions for improving SCADA security one important aspects is the need to design the security knowing the details of the system that is to be secured and knowing the most likely security threats against this system. By using this knowledge the security of the system could be optimized, minimized, and made more cost effective. In this thesis project we have tried to follow this approach throughout the project, by studying SCADA system and the security threats facing these systems. Thus we believe that we have come up with a solution that is both cost effective and minimal. Further suggestions and best practices for securing SCADA system can be found in section 5.2 pages 26.

Even though it is left as future work, exploiting the propriety network protocols to take full advantage of the TPM would be very interesting approach to take. Combining the TPM features and the communication protocols there could be opportunities to further minimize the TPM. Exploring this potential could be very promising and interesting.

10.2 Future work

Despite trying to address this vast problem there are several things that have not been covered due to limitations in resources and time. As mentioned in the analysis section the security analysis is informal method, hence a proper study of attack vectors should be made and the system's performance should be analyzed in detail. In addition, the database system that is supposed to run on the RCM should be studied and suitable algorithms that would fit the system's requirements should be proposed. Finally using the TPM, as a cryptographic engine should also be evaluated, specifically the time constraints and efficiency should be evaluated.

Additionally, other cryptographic techniques, which is less hardware demanding and more secure, should be investigated. The cryptographic techniques were chosen with the primary intention of compatibility with the TCG specification, but finding other techniques that achieves these requirements while consuming less power and area would be interesting. Optimizing the system for speed and size is also an interesting next topic. Another obvious thing to do would be using an updater (or HMI), which has a trusted computing module and the PLC with the embedded trusted computing module and removing the RCM completely or reducing its role in the updating and reprogramming process. Since both the updater and PLC, having a TCM, would be running in a reliable manner there would be no need to have a RCM to verify things but the RCM could be kept as an overseer to monitor and alarm deviation from common trends.

Plus it would be interesting to implement the other two alternatives, which are the solution, which use the RSA server and the solution, which implement the RSA on small processor. It would be interesting to evaluate their performance and cost against the complete hardware implementation proposed here. It would be interesting to study the solutions they present to overcome the major drawbacks in the full hardware implementation.

10.3 Required reflections

Having completed this project I have enhanced my knowledge and understanding regarding scientific writing, arguing a point in writing, supporting my arguments with appropriate references and citations. I have followed the scientific method of doing research, utilizing observations, formulating a hypothesis, reasoning, and experimentation. With this project I have enhanced my engineering skills by formulating unique solutions to obstacles I faced while doing the project.

While doing this project I have greatly enhanced my personal skill of time management, project design, and my ability to model and simulate prototypes. I have used timing charts and Gantt charts to manage my time. I formulated daily tasks, weekly goals, and set project milestones, which were very useful in achieving my goal.

With regards to social and ethical issues, no violation of values and norms occurred. Since most of the experiment was done on simulators it could also be argued that the project development had strong consideration for the environment. It is my personal belief that if vendors implemented the results of this thesis into their PLC, this would contribute to the enhancement and protection of society.

An analysis of the area costs of alternatives has been made. This analysis shows that it is feasible to implement a secure reprogramming solution in PLCs. Given the very high risks to society due to a successful attack, I will also argue that it is clearly cost-effective to include this secure reprogramming into PLCs being used for critical infrastructures. The cost effectiveness of implementing this for all PLCs in non-critical infrastructures remains to be seen in future work.

References

[1] Fortinet Inc., “Securing SCADA Infrastructure,” Fortinet Inc., Sunnyvale, California, USA, White paper WP-SCADA-R1-201010, Oct. 2010.

[2] I. S. of America, *Enterprise-control System Integration: Models and Terminology. Part 1*. The International Society of Automation (ISA), 2000.

[3] Eric Chien, “Stuxnet: A Breakthrough.” 16-Nov-2010.

[4] “Aussie hacker jailed for sewage attacks.” [Online]. Available: http://news.hitb.org/node/4303. [Accessed: 07-Sep-2012].

[5] Tony Smith, “Hacker jailed for revenge sewage attacks,” 31-Oct-2001.

[6] Kevin Poulsen, “Slammer worm crashed Ohio nuke plant network,” 19-Aug-2003. [Online]. Available: http://www.securityfocus.com/news/6767. [Accessed: 29-Jul-2012].

[7] United States Nuclear Regulatory Commission, Office of Nuclear Reactor Regulation, “Effects of Ethernet-based, Non-Safety Related Controls on the Safe and Continued Operation of Nuclear Power Stations,” United States Nuclear Regulatory Commission, Office of Nuclear Reactor Regulation, Washington, DC 20555-0001, USA, NRC Information Notice 2007-15, Apr. 2007.

[8] M. Clayton, “Stuxnet: Ahmadinejad admits cyberweapon hit Iran nuclear program,” *Christian Science Monitor*, 30-Nov-2010.

[9] “Iran confirms Stuxnet found at Bushehr nuclear power plant.” [Online]. Available: http://warincontext.org/2010/09/26/iran-confirms-stuxnet-found-at-bushehr-nuclear-power-plant/. [Accessed: 22-Aug-2012].

[10] N. Falliere, L. O. Murchu, and E. Chien, “W32. stuxnet dossier,” *White paper, Symantec Corp., Security Response*, 2011.

[11] C. A. Theohary, P. K. Kerr, and J. Rollins, *The Stuxnet Computer Worm: Harbinger of an Emerging Warfare Capability - CRS Report*. Congressional Research Service, 2010.

[12] D. E. Sanger, “Obama Ordered Wave of Cyberattacks Against Iran,” *The New York Times*, 01-Jun-2012.

[13] W. Bolton, *Programmable Logic Controllers*. Newnes, 2009.

[14] C. T. Jones, *Programmable Logic Controllers: The Complete Guide to the Technology*. Brilliant-Training, 2006.

[15] R. E. Morley, “United States Patent: 3761893 - DIGITAL COMPUTER,” U.S. Patent 376189325-Sep-1973.

[16] Alison Dunn, “The father of invention: Dick Morley looks back on the 40th anniversary of the PLC,” 12-Sep-2008.

[17] G. R. Clarke, D. Reynders, and E. W. (BSc.), *Practical Modern Scada Protocols: Dnp3, 60870.5 and Related Systems*. Newnes, 2004.

[18] YOKOGAWA electrical corporation, “Reliable and low-cost SCADA-RTU communications using GPRS and DYNAMIC IP,” Jul. 2007.

[19] T. J. Stapko, *Practical Embedded Security: Building Secure Resource-Constrained Systems*. Elsevier, 2007.

[20] G. J. Pottie and W. J. Kaiser, *Principles Of Embedded Networked Systems Design*. Cambridge University Press, 2005.

[21] Whitfield Diffie and Martin E. Hellman, “New Directions in Cryptography.”

[22] "EUR-Lex - 31999L0093 - EN," *Official Journal L 013 , 19/01/2000 P. 0012 - 0020;* [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:HTML. [Accessed: 04-May-2012].
[23] Housley, et. al., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile."
[24] Mozilla.org, "Included Certificate List."
[25] W3Techs, "Usage of SSL certificate authorities for websites."
[26] C. Mitchell, *Trusted Computing*. IET, 2005.
[27] K. J. Knapp, *Cyber-Security and Global Information Assurance: Threat Analysis and Response Solutions*. Idea Group Inc (IGI), 2009.
[28] J. Winter and K. Dietrich, "A hijacker's guide to the LPC bus," *Pre-Proceedings of the*, p. 126, 2011.
[29] S. Pearson, "Trusted Computing: Strengths, Weaknesses and Further Opportunities for Enhancing Privacy," in *Trust Management*, vol. 3477, P. Herrmann, V. Issarny, and S. Shiu, Eds. Springer Berlin / Heidelberg, 2005, pp. 91–117.
[30] S. J. Vaughan-Nichols, "How trustworthy is trusted computing?," *Computer*, vol. 36, no. 3, pp. 18 –20, Mar. 2003.
[31] T. P. M. M. Part, "1—Design Principles, Specification Version 1.2," *Trusted Computing Group*, 2006.
[32] Y. Chen, P. England, M. Peinado, and B. Willman, "High assurance computing on open hardware architectures," *Microsoft re-Rearch Technical Report MSR-TR-2003-20*, 2003.
[33] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," *ARM white paper*, 2004.
[34] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 168–177, 2000.
[35] B. Drury, *The Control Techniques Drives and Controls Handbook*. IET, 2001.
[36] Eric Byres and Justin Lowe, "The Myths and Facts behind Cyber Security Risks for Industrial Control Systems," 2004.
[37] U. S. P. C. on C. I. Protection, *Critical foundations: protecting America's infrastructures : the report of the President's Commission on Critical Infrastructure Protection*. The Commission, 1997.
[38] K. Munro, "SCADA – A critical situation," *Network Security*, vol. 2008, no. 1, pp. 4–6, Jan. 2008.
[39] A. Mahboob and J. Zubairi, "Intrusion avoidance for SCADA security in industrial plants," in *Collaborative Technologies and Systems (CTS), 2010 International Symposium on*, 2010, pp. 447 –452.
[40] P. A. S. Ralston, J. H. Graham, and J. L. Hieb, "Cyber security risk assessment for SCADA and DCS networks," *ISA Trans*, vol. 46, no. 4, pp. 583–594, Oct. 2007.
[41] U.S. Department of Energy National SCADA Test Bed Program, "Top 10 vulnerabilities of control system and their associated mitigation– 2007," Dec. 2006.
[42] Eric Luiijf, "SCADA Security Good Practices for the Drinking Water Sector," TNO, The Hague, TNO-DV 2008 C096, Mar. 2008.

[43] V. I. Nguyen, W. Benjapolakul, and K. Visavateeranon, “A high-speed, low-cost and secure implementation based on embedded ethernet and internet for SCADA systems,” in *SICE, 2007 Annual Conference*, 2007, pp. 1692 –1699.

[44] I. N. Fovino, A. Carcano, and M. Masera, “A Secure and Survivable Architecture for SCADA Systems,” in *Dependability, 2009. DEPEND ’09. Second International Conference on*, 2009, pp. 34 –39.

[45] S. Bagaria, S. B. Prabhakar, and Z. Saquib, “Flexi-DNP3: Flexible distributed network protocol version 3 (DNP3) for SCADA security,” in *Recent Trends in Information Systems (ReTIS), 2011 International Conference on*, 2011, pp. 293 –296.

[46] L. Katzir and I. Schwartzman, “Secure firmware updates for smart grid Devices,” in *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*, 2011, pp. 1 –5.

[47] T. Mander, R. Cheung, and F. Nabhani, “Power system DNP3 data object security using data sets,” *Computers & Security*, vol. 29, no. 4, pp. 487–500, Jun. 2010.

[48] R. Hunt and J. Slay, “Achieving critical infrastructure protection through the interaction of computer security and network forensics,” in *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, 2010, pp. 23 –30.

[49] Norman ASA, “The Norman SCADA Protection system protects against cyber-attacks that target critical SCADA systems.”

[50] S. Valentine and C. Farkas, “Software security: Application-level vulnerabilities in SCADA systems,” in *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*, 2011, pp. 498 –499.

[51] E. B. Fernandez and M. M. Larrondo-Petrie, “Designing Secure SCADA Systems Using Security Patterns,” in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, 2010, pp. 1 –8.

[52] O. Rysavy, J. Rab, P. Halfar, and M. Sveda, “A Formal Authorization Framework for Networked SCADA Systems,” in *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, 2012, pp. 298 –302.

[53] D. K. Holstein and K. Stouffer, “Trust but Verify Critical Infrastructure Cyber Security Solutions,” in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, 2010, pp. 1 –8.

[54] H. Eren and D. Hatipoglu, “Security issues and quality of service in real time wireless PLC/SCADA process control systems,” in *Sensors Applications Symposium, 2008. SAS 2008. IEEE*, 2008, pp. 161 –165.

[55] A. Satoh and T. Inoue, “ASIC-hardware-focused comparison for hash functions MD5, RIPEMD-160, and SHS,” *Integration, the VLSI Journal*, vol. 40, no. 1, pp. 3–10, 2007.

[56] F. Amin, A. Jahangir, and H. Rasifard, “Analysis of public-key cryptography for wireless sensor networks security,” in *Proceedings of world academy of science, engineering and technology*, 2008, vol. 31.

[57] H. Hartig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter, “The Nizza secure-system architecture,” in *Collaborative Computing: Networking, Applications and Worksharing, 2005 International Conference on*, 2005, p. 10–pp.

[58] A. Böttcher, B. Kauer, and H. Härtig, “Trusted computing serving an anonymity service,” *Trusted Computing-Challenges and Applications*, pp. 143–154, 2008.

[59] Y. M. Yussoff and H. Hashim, “Trusted Wireless Sensor Node Platform,” *memory*, vol. 3, p. 6.

[60] K. El Defrawy, D. P. AurAl’lien Francillon, and G. Tsudik, “Smart: Secure and minimal architecture for (establishing dynamic) root of trust,” in *Proceedings of the Network & Distributed System Security Symposium (NDSS), San Diego, CA*, 2012.

[61] K. Dietrich and J. Winter, “Implementation aspects of mobile and embedded trusted computing,” *Trusted Computing*, pp. 29–44, 2009.

[62] V. Costan, L. Sarmenta, M. Van Dijk, and S. Devadas, “The trusted execution module: Commodity general-purpose trusted computing,” *Smart Card Research and Advanced Applications*, pp. 133–148, 2008.

[63] J. H. Burrows, “Secure hash standard,” DTIC Document, 1995.

[64] A. G. Konheim, *Computer security and cryptography*. Jossey-Bass, 2007.

[65] J. Docherty and A. Koelmans, “Hardware Implementation of SHA-1 and SHA-2 Hash Functions,” 2011.

[66] M. J. Wiener, “Performance comparison of public-key cryptosystems,” *RSA Laboratories’ CryptoBytes*, vol. 4, no. 1, p. 1, 1998.

[67] A. Wright, J. Kinast, and J. McCarty, “Low-latency cryptographic protection for SCADA communications,” in *Applied Cryptography and Network Security*, 2004, pp. 263–277.

[68] V. M. Igure, S. A. Laughter, and R. D. Williams, “Security issues in SCADA networks,” *Computers & Security*, vol. 25, no. 7, pp. 498–506, 2006.

[69] A. T. Group and others, “Cryptographic protection of scada communications general recommendations,” *Draft3, AGA Report*, no. 12.

[70] I. NIOS, *Processor Reference Handbook, 2008, Altera Corporation.* .

[71] H. Berger, *Automating with SIMATIC: integrated automation with SIMATIC S7-300/400: controllers, software, programming, data communication, operator control and process monitoring*. Wiley-VCH, 2003.

[72] A. S. Alkalbani, T. Mantoro, and A. O. M. Tap, “Comparison between RSA hardware and software implementation for WSNs security schemes,” in *2010 International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, 2010, pp. E84 –E89.

[73] B. C. Neuman and T. Ts’o, “Kerberos: an authentication service for computer networks,” *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33 –38, Sep. 1994.

Appendix A.   VHDL of implementation

The Execution Engine

```
        --
*********************************************************************************************
**
-- Execution Engine for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
*********************************************************************************************
**

library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use work.my_library.all;
--use work.my_library.MEMORY_RW.all;

-- naming "_n" implies active low
--
ENTITY ExecutionEngine IS
  GENERIC (      NumberOfMasters: int :=1);
  PORT (OPCODE  :IN STD_LOGIC_VECTOR (39 DOWNTO 0);-- load the opcode from the execution
memory
    RESET_n     : IN STD_LOGIC;
    CLK         : IN STD_LOGIC;

    --OPCODE_OUT  : OUT  STD_LOGIC_VECTOR (39 DOWNTO 0);
    -------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------
    SLCT_DEVICE :OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    -------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------
    EEROM_ADDRS : OUT STD_LOGIC_VECTOR (7 downto 0);-- address from operands
    --------------COMMUNICATION WITH MAIN MEMORY
    MM_WREN     : OUT STD_LOGIC;
    MM_BRST_SIZE: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
    MM_RAM_ADDRS: OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    MM_MEM_ADDRS: OUT STD_LOGIC_VECTOR (19 DOWNTO 0);
    MM_DONE     : IN  STD_LOGIC;
    -------------------------COOMUNICATION WITH MAIN MEMORY END------------------------------
---------------------------------------------------------------------------------------------
    COMP_LENGTH :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_ADDRS2 :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_ADDRS1 :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_DONE   :IN STD_LOGIC;
    COMP_RST    :IN STD_LOGIC;
    -------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------
    SHA_START   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_END     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_RSLT    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_DONE    : IN STD_LOGIC;
    -------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------
    RSA_START   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_END     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_RSLT    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_DONE    : IN STD_LOGIC;
    -------------------------------------------------------------------------------------
    -------------------------------------------------------------------------------------
    RNG_SEED    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_LENGTH  : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_OUT_ADD : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_DONE    : IN STD_LOGIC  );
END ExecutionEngine;
ARCHITECTURE EE OF ExecutionEngine IS
```

```
SIGNAL OPERATION :  STD_LOGIC_VECTOR ( 39 DOWNTO 0);

SIGNAL nxt_addr : STD_LOGIC_VECTOR (7 DOWNTO 0):=(OTHERS=>'0');
SIGNAL COMP_FLAG : STD_LOGIC;
SIGNAL BOOT_FLAG : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL BOOT_REG  : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL DONE_m    : STD_LOGIC;
BEGIN
 OPERATION <= OPCODE;
 SLCT_DEVICE<= OPERATION(39 DOWNTO 37);
 --DONE_m<= DONE;
 SET_BOOT_REG: PROCESS (CLK, RESET_n, BOOT_FLAG)
 BEGIN
   IF(RISING_EDGE(CLK)AND RESET_n= '1')THEN
    CASE BOOT_FLAG IS
    WHEN "000" =>
      BOOT_REG<=  BOOT_REG(7 DOWNTO 1)& '1';
    WHEN "001" =>
      BOOT_REG<= BOOT_REG(7 DOWNTO 2)&'1'& BOOT_REG(0 DOWNTO 0);
    WHEN OTHERS =>
      BOOT_REG<= BOOT_REG;
    END CASE;
   ELSIF (RESET_N='0')THEN
    BOOT_REG<= (OTHERS=>'0');
   END IF;
 END PROCESS SET_BOOT_REG;


DECODE: PROCESS(CLK, RESET_n)
VARIABLE rtn_addr : STD_LOGIC_VECTOR (7 DOWNTO 0):= (OTHERS=>'0');
VARIABLE JUMP_FLAG : STD_LOGIC;
VARIABLE REG  : STD_LOGIC_VECTOR (7 DOWNTO 0):=(OTHERS=>'0');
VARIABLE ADDRS_r: STD_LOGIC_VECTOR (15 DOWNTO 0):= (OTHERS=>'0');
VARIABLE ADDRS_w: STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS=>'0');

BEGIN
  IF(RESET_n = '1' AND RISING_EDGE(CLK)AND DONE_m='1') THEN  -- not reseted
    CASE OPERATION (39 DOWNTO 37)  IS -- DECODE THE EXECUTION CODE
    WHEN "000" =>
      JUMP_FLAG:='1';
    WHEN "001" => -- read /write to/from ram to main memory
      MM_WREN    <=OPERATION (36);
      MM_BRST_SIZE <=OPERATION(35 DOWNTO 30);
      MM_RAM_ADDRS <=OPERATION(29 DOWNTO 20);
      MM_MEM_ADDRS <=OPERATION(19 DOWNTO 0);
      DONE_m<=MM_DONE;
     -- // MIGHT BE WORTH CHECKING BEFOR USE // MEMORY_RW (OPCODE (29 DOWNTO 28),  OPCODE (27
DOWNTO 25), GRANT_n , CLK , nxt ,  ADDRS_r ,  ADDRS_w);-- ADDRS_r/w CAN BE EXCHANGED AS
DESIRED
    WHEN "010" => -- compare two MEMEORY BLOCKES  bit values
      COMP_LENGTH <= OPERATION (29 DOWNTO 20);
      COMP_ADDRS2 <= OPERATION (19 DOWNTO 10);
      COMP_ADDRS1 <= OPERATION (9 DOWNTO 0);
      DONE_m<= COMP_DONE;
    WHEN "011" => -- SHA-1 USED TO HASH DATA RECIVE MEMORY RANGE WITH FORMAT START ADDRESS AND
END ADDRESS
      SHA_START <= OPERATION (29 DOWNTO 20);
      SHA_END   <= OPERATION (19 DOWNTO 10);
      SHA_RSLT  <= OPERATION (9 DOWNTO 0);
      DONE_m       <= SHA_DONE;
    WHEN "100" =>  -- RSA-512 key
      RSA_START <= OPERATION (29 DOWNTO 20);
      RSA_END   <= OPERATION (19 DOWNTO 10);
      RSA_RSLT  <= OPERATION (9 DOWNTO 0);
      DONE_m       <= RSA_DONE;
    WHEN "101" =>  -- RNG
      RNG_SEED  <=OPERATION (29 DOWNTO 20);
      RNG_LENGTH <=OPERATION (19 DOWNTO 10);
      RNG_OUT_ADD <= OPERATION (9 DOWNTO 0);
      DONE_m       <=  RNG_DONE;
    WHEN "110" => -- WRITE KEY RAM
      MM_WREN    <=OPERATION (36);
      MM_BRST_SIZE <=OPERATION(35 DOWNTO 30);
      MM_RAM_ADDRS <=OPERATION(29 DOWNTO 20);
      MM_MEM_ADDRS <=OPERATION(19 DOWNTO 0);
      DONE_m<=MM_DONE;

    WHEN OTHERS => DONE_m <= '1';
    END CASE;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
--ADDRS_GNR : PROCESS (CLK, RESET_n, OPERATION,DONE_m )
--VARIABLE nxt_addr : STD_LOGIC_VECTOR (7 DOWNTO 0);
--VARIABLE rtn_addr : STD_LOGIC_VECTOR (7 DOWNTO 0);
--VARIABLE JUMP_FLAG : STD_LOGIC;
--BEGIN
--  IF(RESET_n = '1' AND CLK ='1' AND DONE_m='1') THEN  -- not reseted AND OPERATION WAS DONE
    IF(JUMP_FLAG= '1') THEN
      JUMP_FLAG := '0';
      CASE OPERATION (36 DOWNTO 30) IS
      WHEN "0000000"=> -- SIMPLE JUMP
        nxt_addr <= nxt_addr + OPERATION(7 DOWNTO 0);--  since opcode is 40bit long we read
every five bits
      WHEN "0000001" => -- CONDITIONAL JUMP -- JUMP IF NOT OKAY
        IF (COMP_DONE = '1' AND COMP_FLAG='0')THEN -- COMPARISON
          nxt_addr <= nxt_addr + OPERATION (7 DOWNTO 0);
        ELSE
          nxt_addr <= nxt_addr + x"01";-- CONDITON OKAY
          BOOT_FLAG <= OPERATION (29 DOWNTO 27); --SINCE BOOT FLAG IS INITALIZE TO ZERO WE
ONLY NEED TO SET IT BACK TO 1
        END IF;
      WHEN "0000010"=> -- FORKING BOOT
        IF (COMP_DONE = '1' AND COMP_FLAG='0')THEN -- COMPARISON
          nxt_addr <= nxt_addr + OPERATION (7 DOWNTO 0);-- HERE BOOT FLAG IS LEFT ZERO
        ELSE
          nxt_addr <= nxt_addr + OPERATION (15 DOWNTO 8);-- CONDITON OKAY
          BOOT_FLAG <= OPERATION (29 DOWNTO 27); --SINCE BOOT FLAG IS INITALIZE TO ZERO WE
ONLY NEED TO SET IT BACK TO 1
        END IF;
      WHEN "0000011" => --- JUMP WITH RETURN ADDRESS
        rtn_addr:= nxt_addr + x"01";
        nxt_addr <= nxt_addr + OPERATION (7 DOWNTO 0);
      WHEN "0000100" => -- JUMP TO RETN ADDRS
        nxt_addr <= rtn_addr;
        rtn_addr := (OTHERS =>'0');
      WHEN OTHERS =>
        nxt_addr <= nxt_addr + x"01";
      END CASE;
    ELSE
      nxt_addr <= nxt_addr + x"01";
    END IF;-- for jump
    DONE_m<= '1';
    EEROM_ADDRS<= nxt_addr;
--   END IF;
--
--END PROCESS ADDRS_GNR;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
  ELSE -- reseted
    DONE_m <='1';
  END IF; --END FOR RESET AND CLK
END PROCESS DECODE;
END EE;
```

Avalon Master Interface

```
--
**********************************************************************************************
**
-- Top TCM I/O for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
**********************************************************************************************
**

library IEEE, work;
```

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
--use work.my_library.all;
--use work.my_library.MEMORY_RW.all;

ENTITY AVALON_MM_BRUST_MASTER IS
  GENERIC (MM_BASE_ADDRS : STD_LOGIC_VECTOR(11 DOWNTO 0):= x"000" );
  PORT (CLK : IN STD_LOGIC;
    RESET_n : IN STD_LOGIC;
    MM_WREN     : IN STD_LOGIC;
    MM_BRST_SIZE: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
    MM_RAM_ADDRS: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
    MM_MEM_ADDRS: IN STD_LOGIC_VECTOR (19 DOWNTO 0);
    MM_DONE     : OUT STD_LOGIC;

    RAM_ADDRS :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RAM_DATA  :OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
    RAM_Q     :IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    RAM_WREN  :OUT STD_LOGIC;


    MASTER_BEGINTRANSFER: OUT STD_LOGIC;
    -- MASTER_BYTEENABLE : OUT STD_LOGIC_VECTOR (3 DOWNTO 0); -- IGNORED CAUSE DEFAULT IS 32
BIT

    MASTER_RD :OUT STD_LOGIC;
    MASTER_RD_ADDRS:OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
    MASTER_RD_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO  0);
    MASTER_RD_DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    MASTER_RD_DATA_VALID: IN STD_LOGIC;
    MASTER_RD_WAIT_RQST : IN STD_LOGIC;

    MASTER_WR :OUT STD_LOGIC;
    MASTER_WR_ADDRS: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    MASTER_WR_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
    MASTER_WR_DATA: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    MASTER_WR_WAIT_RQST :IN STD_LOGIC
     );
END ENTITY ;

ARCHITECTURE MM_RW OF AVALON_MM_BRUST_MASTER IS
SIGNAL RD_DATA : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL WR_DATA : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL NXT_RAM_ADDRS_R: STD_LOGIC_VECTOR (9 DOWNTO 0);
SIGNAL NXT_RAM_ADDRS_W: STD_LOGIC_VECTOR (9 DOWNTO 0);
SIGNAL BURST_STARTED_R: STD_LOGIC:='0';
SIGNAL BURST_STARTED_W: STD_LOGIC:='0';
SIGNAL COUNT :STD_LOGIC_VECTOR (5 DOWNTO 0):= "000000";
SIGNAL MM_DONE_R:STD_LOGIC;
SIGNAL MM_DONE_W:STD_LOGIC;
SIGNAL MASTER_BEGINTRANSFER_R: STD_LOGIC;
SIGNAL MASTRR_BEGINTRANSFER_W: STD_LOGIC;

BEGIN
RAM_DATA<=MASTER_RD_DATA;
MASTER_WR_DATA<= RAM_Q;
RAM_WREN<= NOT MM_WREN; -- WHEN NOT MM_WREN I.E RD_MAIN_MEMEORY AND WRITE RAM
MASTER_WR<= MM_WREN;
MASTER_RD<= NOT MM_WREN;
MASTER_WR_BURST_COUNT <= MM_BRST_SIZE WHEN RESET_n='1' ELSE "000000";
MASTER_RD_BURST_COUNT <= MM_BRST_SIZE WHEN RESET_n='1' ELSE "000000";
MM_DONE<= '1' WHEN MM_DONE_R='1'OR MM_DONE_W='1'ELSE '0';
MASTER_BEGINTRANSFER <= MASTRR_BEGINTRANSFER_W WHEN MM_WREN='1' ELSE MASTER_BEGINTRANSFER_R;
RAM_ADDRS<= NXT_RAM_ADDRS_W WHEN MM_WREN='1' ELSE NXT_RAM_ADDRS_R;
RD_MASTER: PROCESS (CLK, RESET_n, MASTER_RD_DATA_VALID,MASTER_RD_WAIT_RQST, MM_WREN)

BEGIN
  IF(RESET_n='0') THEN
    NXT_RAM_ADDRS_R<="0000000000";
    BURST_STARTED_R<='0';
  ELSIF (RESET_n='1' AND (RISING_EDGE(CLK) ) AND MM_WREN = '0') THEN
    --RAM_ADDRS<= NXT_RAM_ADDRS_R;
    IF (BURST_STARTED_R ='0' AND MASTER_RD_WAIT_RQST='1') THEN
```

```
      MASTER_BEGINTRANSFER_R <='1';
      MASTER_RD_ADDRS<= MM_BASE_ADDRS & MM_MEM_ADDRS;
    ELSIF (BURST_STARTED_R ='0' AND MASTER_RD_WAIT_RQST='0') THEN
      BURST_STARTED_R<= '1';
      NXT_RAM_ADDRS_R<= MM_RAM_ADDRS;
      MASTER_BEGINTRANSFER_R<='0';
    ELSIF (BURST_STARTED_R='1' AND MASTER_RD_WAIT_RQST='1') THEN
      NXT_RAM_ADDRS_R<= NXT_RAM_ADDRS_R;
    ELSE
      IF(MASTER_RD_DATA_VALID= '1')THEN
        NXT_RAM_ADDRS_R<= NXT_RAM_ADDRS_R + "0000000001";
      END IF;
    END IF;
    IF(NXT_RAM_ADDRS_R = (MM_RAM_ADDRS +( "0000" & MM_BRST_SIZE))) THEN
      MM_DONE_R<= '1';
    ELSE
      MM_DONE_R<='0';
    END IF;
  END IF;
END PROCESS RD_MASTER;

WR_MASTER:  PROCESS (CLK, RESET_n, MASTER_WR_WAIT_RQST, MM_WREN)
BEGIN
  IF(RESET_n='0') THEN
  NXT_RAM_ADDRS_W<="0000000000";
  BURST_STARTED_W<='0';
  ELSIF (RESET_n='1' AND RISING_EDGE(CLK) AND MM_WREN = '1') THEN
    --RAM_ADDRS<= NXT_RAM_ADDRS_W;
    IF (BURST_STARTED_W ='0' AND MASTER_WR_WAIT_RQST='1') THEN
      MASTRR_BEGINTRANSFER_W <='1';
      MASTER_WR_ADDRS<= MM_BASE_ADDRS & MM_MEM_ADDRS;
    ELSIF (BURST_STARTED_W ='0' AND MASTER_WR_WAIT_RQST='0') THEN
      BURST_STARTED_W<= '1';
      NXT_RAM_ADDRS_W<= MM_RAM_ADDRS;
      MASTRR_BEGINTRANSFER_W<='0';
    ELSIF (BURST_STARTED_W='1' AND MASTER_WR_WAIT_RQST='1') THEN
      NXT_RAM_ADDRS_W<= NXT_RAM_ADDRS_W;
    ELSE
        NXT_RAM_ADDRS_W<= NXT_RAM_ADDRS_W + "0000000001";
    END IF;
    IF(NXT_RAM_ADDRS_W = (MM_RAM_ADDRS +( "0000" & MM_BRST_SIZE))) THEN
      MM_DONE_W<= '1';
    ELSE
      MM_DONE_W<='0';
    END IF;

  END IF;
END PROCESS WR_MASTER;
END MM_RW;
```

IO module

```
--
*********************************************************************************************
**
-- Top TCM I/O for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
*********************************************************************************************
**

library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use  work.my_library.all;
--use work.my_library.MEMORY_RW.all;

ENTITY TOP_TCM IS
GENERIC (BASE_ADDRS: STD_LOGIC_VECTOR(11 DOWNTO 0):= X"000");
PORT ( CLK_IN : IN STD_LOGIC;
       CLK_RSA: IN STD_LOGIC;
       RESET_n : IN STD_LOGIC;
       MASTER_BEGINTRANSFER: OUT STD_LOGIC;
       -- MASTER_BYTEENABLE : OUT STD_LOGIC_VECTOR (3 DOWNTO 0); -- IGNORED CAUSE DEFAULT IS
32 BIT

       MASTER_RD :OUT STD_LOGIC;
       MASTER_RD_ADDRS:OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
       MASTER_RD_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO  0);
       MASTER_RD_DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
       MASTER_RD_DATA_VALID: IN STD_LOGIC;
       MASTER_RD_WAIT_RQST : IN STD_LOGIC;

       MASTER_WR :OUT STD_LOGIC;
       MASTER_WR_ADDRS: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       MASTER_WR_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
       MASTER_WR_DATA: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
       MASTER_WR_WAIT_RQST :IN STD_LOGIC
       );
END ENTITY;

ARCHITECTURE rtl OF TOP_TCM IS
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT ExecutionEngine IS
  GENERIC (     NumberOfMasters: int :=1);
  PORT (OPCODE  :IN STD_LOGIC_VECTOR (39 DOWNTO 0);-- load the opcode from the execution
memory
    RESET_n     : IN STD_LOGIC;
    CLK         : IN STD_LOGIC;

    --OPCODE_OUT  : OUT  STD_LOGIC_VECTOR (39 DOWNTO 0);
    ------------------------------------------------------------------------------------------
    ------------------------------------------------------------------------------------------
    SLCT_DEVICE :OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    ------------------------------------------------------------------------------------------
    ------------------------------------------------------------------------------------------
    EEROM_ADDRS : OUT STD_LOGIC_VECTOR (7 downto 0);-- address from operands
    --------------COMMUNICATION WITH MAIN MEMORY
    MM_WREN     : OUT STD_LOGIC;
    MM_BRST_SIZE: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
    MM_RAM_ADDRS: OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    MM_MEM_ADDRS: OUT STD_LOGIC_VECTOR (19 DOWNTO 0);
    MM_DONE     : IN  STD_LOGIC;
    -------------------------COOMUNICATION WITH MAIN MEMORY END-------------------------------
    ------------------------------------------------------------------------------------------
    COMP_LENGTH :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_ADDRS2 :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_ADDRS1 :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    COMP_DONE   :IN STD_LOGIC;
    COMP_RST    :IN STD_LOGIC;
    ------------------------------------------------------------------------------------------
    ------------------------------------------------------------------------------------------
    SHA_START   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_END     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_RSLT    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    SHA_DONE    : IN STD_LOGIC;
    ------------------------------------------------------------------------------------------
    ------------------------------------------------------------------------------------------
    RSA_START   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_END     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_RSLT    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RSA_DONE    : IN STD_LOGIC;
    ------------------------------------------------------------------------------------------
    ------------------------------------------------------------------------------------------
    RNG_SEED    : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_LENGTH  : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_OUT_ADD : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RNG_DONE    : IN STD_LOGIC   );
END COMPONENT ;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT AVALON_MM_BRUST_MASTER IS
  GENERIC (MM_BASE_ADDRS : STD_LOGIC_VECTOR :=x"000");
  PORT (CLK : IN STD_LOGIC;
    RESET_n : IN STD_LOGIC;
    MM_WREN     : IN STD_LOGIC;
    MM_BRST_SIZE: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
    MM_RAM_ADDRS: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
    MM_MEM_ADDRS: IN STD_LOGIC_VECTOR (19 DOWNTO 0);
    MM_DONE     : OUT STD_LOGIC;

    RAM_ADDRS :OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
    RAM_DATA  :OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
    RAM_Q     :IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    RAM_WREN  :OUT STD_LOGIC;


    MASTER_BEGINTRANSFER: OUT STD_LOGIC;
    -- MASTER_BYTEENABLE : OUT STD_LOGIC_VECTOR (3 DOWNTO 0); -- IGNORED CAUSE DEFAULT IS 32
BIT

    MASTER_RD :OUT STD_LOGIC;
    MASTER_RD_ADDRS:OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
    MASTER_RD_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO  0);
    MASTER_RD_DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    MASTER_RD_DATA_VALID: IN STD_LOGIC;
    MASTER_RD_WAIT_RQST : IN STD_LOGIC;

    MASTER_WR :OUT STD_LOGIC;
    MASTER_WR_ADDRS: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    MASTER_WR_BURST_COUNT: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
    MASTER_WR_DATA: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
    MASTER_WR_WAIT_RQST :IN STD_LOGIC
     );
END COMPONENT;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------

COMPONENT EE_instruction_ROM IS
  PORT
  (
    address             : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    clock               : IN STD_LOGIC  := '1';
    q            : OUT STD_LOGIC_VECTOR (39 DOWNTO 0)
  );
END COMPONENT ;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT SHA1_wrap IS
  PORT (SHA_START   : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_END     : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_DONE    : OUT STD_LOGIC;
        SHA_RSLT    : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RESET_n     : IN STD_LOGIC;
        DATA_ADDRS  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0);
        DATA        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        WREN        : OUT STD_LOGIC;
        --OUT_ADDRS   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_OUT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        CLK         : STD_LOGIC);
  END COMPONENT;

----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT Volatile_memory_RAM IS
  PORT
  (
    address             : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
    clock               : IN STD_LOGIC  := '1';
    data        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    wren        : IN STD_LOGIC ;
    q           : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
  );
END COMPONENT ;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT Comparator IS
  GENERIC (NumberOfMasters: int :=1);
  PORT( COMP_LENGTH : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        COMP_ADDRS2 : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        COMP_ADDRS1 : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        COMP_DONE   : OUT STD_LOGIC;
        COMP_RSLT   : OUT STD_LOGIC;
        RESET_n     : IN STD_LOGIC;
        DATA_ADDRS  : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
        DATA        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        WREN        : OUT STD_LOGIC;
        COMP_OUT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        CLK         : IN STD_LOGIC);
END COMPONENT ;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
COMPONENT RSA_wrap IS
  PORT (RSA_START   : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RSA_END     : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RSA_DONE    : OUT STD_LOGIC;
        RSA_RSLT    : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RESET_n     : IN STD_LOGIC;
        DATA_ADDRS  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0);
        DATA        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        WREN        : OUT STD_LOGIC;
        RSA_OUT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        RSA_RAM_ADDRS: IN STD_LOGIC_VECTOR (6 DOWNTO 0);-- ADDRESS TO WRITE THE KES ADDRESS
RANGE 0000000 TO 0011111 ADDRESS THE KEYS
        RSA_RAM_DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);--WHILE 0100000 TO 0111111 ADDRESS THE
M VALUES and 1000000 TO 1011111 THE R_C
        RSA_RAM_WREN: IN STD_LOGIC;
        RSA_CLK     : IN STD_LOGIC;
        CLK         : IN STD_LOGIC);
  END COMPONENT;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
SIGNAL ADDRS_m : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL CLK_m   : STD_LOGIC;
SIGNAL Q_m     : STD_LOGIC_VECTOR (39 DOWNTO 0);
SIGNAL SHA_CABLE : STD_LOGIC_VECTOR(31 DOWNTO 0);-- START_ADDRES 9TO0, END_ADDRS 19TO10,
OUT_ADDRESS 29TO 20, RESET_n AT 30, DONE AT 31
SIGNAL RSA_CABLE : STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL MEM_CABLE : STD_LOGIC_VECTOR (37 DOWNTO 0);----MM_WREN (0)MM_BRST_SIZE(6
TO1)MM_RAM_ADDRS(16 TO7)MM_MEM_ADDRS(36 TO 17)MM_DONE(37)
SIGNAL RAM_CABLE : STD_LOGIC_VECTOR (74 DOWNTO 0);-- 9 DOWNTO 0 ADDRESS, FROM 41 DOWNTO 10
INDATA WRITE , FROM 73 DOWNTO 42 OUTDATA, 74 WREN
SIGNAL COMP_CABLE : STD_LOGIC_VECTOR(32 DOWNTO 0);-- 9 TO 0 LENGTH 19 TO 10 ADDRES2 29 TO 20
ADDRS1 30 DONE 31 RESET 32 RESULT
SIGNAL SHA_TO_RAM: STD_LOGIC_VECTOR (74 DOWNTO 0);
SIGNAL RAM_TO_MEM: STD_LOGIC_VECTOR (74 DOWNTO 0); -- 9 DOWNTO 0 ADDRESS, FROM 41 DOWNTO 10
INDATA WRITE , FROM 73 DOWNTO 42 OUTDATA, 74 WREN
SIGNAL MEM_TO_RSA_RAM : STD_LOGIC_VECTOR (74 DOWNTO 0);
SIGNAL MEM_DATA  : STD_LOGIC_VECTOR (74 DOWNTO 0);
SIGNAL RSA_TO_RAM: STD_LOGIC_VECTOR (74 DOWNTO 0);
SIGNAL COMP_TO_RAM: STD_LOGIC_VECTOR(74 DOWNTO 0);
SIGNAL SLCT_DEVICE : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL DEVICE : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL RAM_SLCT_m :STD_LOGIC;
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
BEGIN
WITH SLCT_DEVICE SELECT
  DEVICE <= "0010" WHEN "010",--COMP
            "0100" WHEN "011",-- SHA
            "1000" WHEN "100",-- RSA
            "0001" WHEN OTHERS;--EE
SHA_CABLE(31)<= DEVICE (2);
RSA_CABLE(31)<= DEVICE (1);
WITH SLCT_DEVICE SELECT
  RAM_CABLE <= SHA_TO_RAM WHEN "011",--SHA
               RSA_TO_RAM WHEN "100",-- RSA
               RAM_TO_MEM  WHEN OTHERS; --MEM
WITH RAM_SLCT_m SELECT
  MEM_DATA <= MEM_TO_RSA_RAM WHEN '0',
             RAM_TO_MEM     WHEN OTHERS;
MEM_DATA<= MEM_TO_RSA_RAM WHEN SLCT_DEVICE= "110" ELSE RAM_TO_MEM;
```

```
ROM: EE_instruction_ROM PORT MAP (address =>ADDRS_m ,clock =>CLK_IN , q => Q_m  );
RAM: Volatile_memory_RAM PORT MAP (address =>RAM_CABLE(9 DOWNTO 0),clock =>CLK_IN, data   =>
RAM_CABLE(41 DOWNTO 10),wren     => RAM_CABLE(74),q => RAM_CABLE (73 DOWNTO 42));

EE: ExecutionEngine PORT MAP (OPCODE => Q_m,
  RESET_n    =>RESET_n,-- : IN STD_LOGIC,
  CLK        =>CLK_IN,-- : IN STD_LOGIC,

  --OPCODE_OUT  : OUT  STD_LOGIC_VECTOR (39 DOWNTO 0),
  -----------------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------------
  SLCT_DEVICE =>SLCT_DEVICE,
  -----------------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------------
  EEROM_ADDRS => ADDRS_m,
  --------------COMMUNICATION WITH MAIN MEMORY -----------------------------
  MM_WREN     =>MEM_CABLE(0),
  MM_BRST_SIZE =>MEM_CABLE(6 DOWNTO 1),
  MM_RAM_ADDRS =>MEM_CABLE(16 DOWNTO 7),
  MM_MEM_ADDRS =>MEM_CABLE(36 DOWNTO 17),
  MM_DONE     =>MEM_CABLE(37),
  ------------------------COOMUNICATION WITH MAIN MEMORY END-------------
  -----------------------------------------------------------------------------------------------
  COMP_LENGTH =>COMP_CABLE(9 DOWNTO 0),
  COMP_ADDRS2 =>COMP_CABLE (19 DOWNTO 10),
  COMP_ADDRS1 =>COMP_CABLE (29 DOWNTO 20),
  COMP_DONE   => COMP_CABLE (30),
  COMP_RST    => COMP_CABLE(32),
  -----------------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------------
  SHA_START   => SHA_CABLE(9 DOWNTO 0),
  SHA_END     => SHA_CABLE(19 DOWNTO 10),
  SHA_RSLT    => SHA_CABLE(29 DOWNTO 20),
  SHA_DONE    => SHA_CABLE(30),
  -----------------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------------
  RSA_START  =>RSA_CABLE (9 DOWNTO 0),-- : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0),
  RSA_END    =>RSA_CABLE(19 DOWNTO 10),--  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO
0),
  RSA_RSLT   =>RSA_CABLE (29 DOWNTO 20),--  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO
0),
  RSA_DONE    => RSA_CABLE(30),--  : IN STD_LOGIC,
  -----------------------------------------------------------------------------------------------
  -----------------------------------------------------------------------------------------------
  RNG_SEED    =>OPEN,
  RNG_LENGTH  =>OPEN,
  RNG_OUT_ADD =>OPEN,
  RNG_DONE    =>'0'  );

SHA1: SHA1_wrap PORT MAP (SHA_START   => SHA_CABLE(9 DOWNTO 0),
      SHA_END     => SHA_CABLE(19 DOWNTO 10),
      SHA_DONE    => SHA_CABLE(30),
      SHA_RSLT    => SHA_CABLE (29 DOWNTO 20),
      RESET_n     => SHA_CABLE (31),
      DATA_ADDRS  => SHA_TO_RAM(9 DOWNTO 0),
      DATA        => SHA_TO_RAM (73 DOWNTO 42),
      WREN        => SHA_TO_RAM (74),
      --OUT_ADDRS   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
      SHA_OUT     => SHA_TO_RAM (41 DOWNTO 10),
      CLK          => CLK_IN);

RSA : RSA_wrap PORT MAP (RSA_START  => RSA_CABLE (9 DOWNTO 0),
      RSA_END    => RSA_CABLE (19 DOWNTO 10),
      RSA_DONE   => RSA_CABLE (30),
      RSA_RSLT   => RSA_CABLE (29 DOWNTO 20),
      RESET_n    => RSA_CABLE(31),
      DATA_ADDRS => RSA_TO_RAM (9 DOWNTO 0),
      DATA       => RSA_TO_RAM (73 DOWNTO 42),
      WREN       => RSA_TO_RAM (74),
      RSA_OUT    => RSA_TO_RAM (41 DOWNTO 10),
      RSA_RAM_ADDRS => MEM_TO_RSA_RAM(6 DOWNTO 0),-- ADDRESS TO WRITE THE KES ADDRESS RANGE
0000000 TO 0011111 ADDRESS THE KEYS
      RSA_RAM_DATA  => MEM_TO_RSA_RAM(41 DOWNTO 10),--WHILE 0100000 TO 0111111 ADDRESS THE M
VALUES and 1000000 TO 1011111 THE R_C
```

```
      RSA_RAM_WREN  => MEM_TO_RSA_RAM(74), -- 9 DOWNTO 0 ADDRESS, FROM 41 DOWNTO 10 INDATA
WRITE , FROM 73 DOWNTO 42 OUTDATA, 74 WREN
      RSA_CLK     => CLK_RSA,
      CLK         => CLK_IN);

COMPR: Comparator
  GENERIC MAP (NumberOfMasters => 1)
  PORT MAP( COMP_LENGTH => COMP_CABLE (9 DOWNTO 0),
        COMP_ADDRS2 => COMP_CABLE (19 DOWNTO 10),
        COMP_ADDRS1 => COMP_CABLE (29 DOWNTO 20),
        COMP_DONE   => COMP_CABLE(30),
        COMP_RSLT   => COMP_CABLE (32),
        RESET_n     => COMP_CABLE (31),
        DATA_ADDRS  => COMP_TO_RAM(9 DOWNTO 0),
        DATA        => COMP_TO_RAM(73 DOWNTO 42),
        WREN        => COMP_TO_RAM(74),
        COMP_OUT     => COMP_TO_RAM (41 DOWNTO 10),
        CLK         => CLK_IN);
MEM :AVALON_MM_BRUST_MASTER
  GENERIC MAP (MM_BASE_ADDRS => x"000")
  PORT MAP (CLK =>CLK_IN,
    RESET_n => RESET_n,
    MM_WREN     =>MEM_CABLE(0),
    MM_BRST_SIZE =>MEM_CABLE(6 DOWNTO 1),
    MM_RAM_ADDRS =>MEM_CABLE(16 DOWNTO 7),
    MM_MEM_ADDRS =>MEM_CABLE(36 DOWNTO 17),
    MM_DONE     =>MEM_CABLE(37),
    -- 9 DOWNTO 0 ADDRESS, FROM 41 DOWNTO 10 INDATA WRITE , FROM 73 DOWNTO 42 OUTDATA, 74 WREN
    RAM_ADDRS   =>MEM_DATA(9 DOWNTO 0),
    RAM_DATA    =>MEM_DATA(41 DOWNTO 10),
    RAM_Q       =>MEM_DATA(73 DOWNTO 42),
    RAM_WREN    =>MEM_DATA(74),

    MASTER_BEGINTRANSFER=> MASTER_BEGINTRANSFER,
    -- MASTER_BYTEENABLE : OUT STD_LOGIC_VECTOR (3 DOWNTO 0); -- IGNORED CAUSE DEFAULT IS 32
BIT

    MASTER_RD => MASTER_RD,
    MASTER_RD_ADDRS=> MASTER_RD_ADDRS,
    MASTER_RD_BURST_COUNT=>MASTER_RD_BURST_COUNT,
    MASTER_RD_DATA=> MASTER_RD_DATA,
    MASTER_RD_DATA_VALID=> MASTER_RD_DATA_VALID,
    MASTER_RD_WAIT_RQST => MASTER_RD_WAIT_RQST,

    MASTER_WR =>MASTER_WR,
    MASTER_WR_ADDRS=> MASTER_WR_ADDRS,
    MASTER_WR_BURST_COUNT=> MASTER_WR_BURST_COUNT,
    MASTER_WR_DATA=>MASTER_WR_DATA,
    MASTER_WR_WAIT_RQST => MASTER_WR_WAIT_RQST
     );
END ARCHITECTURE;
```

Access controller

```
--
*********************************************************************************************
**
-- Memory controller for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
*********************************************************************************************
**
library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use work.my_library.all;
--use work.my_library.small_int;

-- naming "_n" implies active low so in WR_n write is when WR_n=0
--
ENTITY MemoryController is
```

```
                        GENERIC (
                                                Memory1End: std_logic_vector
(MemoryAddressSize downto 0):= CONV_STD_LOGIC_VECTOR(128,(MemoryAddressSize + 1));

                                                Memory1Start: std_logic_vector
(MemoryAddressSize downto 0):= CONV_STD_LOGIC_VECTOR(0,(MemoryAddressSize + 1));
                                                Memory2End: std_logic_vector
(MemoryAddressSize downto 0):= CONV_STD_LOGIC_VECTOR(1024,(MemoryAddressSize + 1));
                                                Memory2Start: std_logic_vector
(MemoryAddressSize downto 0):= CONV_STD_LOGIC_VECTOR(128,(MemoryAddressSize + 1));
                                                Memory3End: std_logic_vector
(MemoryAddressSize downto 0):= CONV_STD_LOGIC_VECTOR(2048,(MemoryAddressSize + 1));
            Memory3Start: std_logic_vector (MemoryAddressSize downto 0):=
CONV_STD_LOGIC_VECTOR(128,(MemoryAddressSize + 1)));
                        PORT(
                                         TPMState    : IN  STD_LOGIC_VECTOR (2 downto 0);--
the state of the tpm are 00-TPM boot,01-update, 10-SEC-exe,11-exit
                                         BUSREQ_n    : IN  STD_LOGIC_VECTOR (NumberOfMasters
downto 0);-- which master or slave has requested bus like requester id with the TPM with ID=0;
                                         TRANSType         : IN STD_LOGIC_VECTOR (1 downto
0);-- type of transfer weather it is 00-IDEL; 01-BUSY; 10-NONSEQ; 11-SEQ
                 ADDRS       : IN  STD_LOGIC_VECTOR (MemoryAddressSize downto 0);-- the
address of the memory requested
                                         SIZE                : IN STD_LOGIC_VECTOR(2 downto
0);-- size of the transfer need to be checked for brust operation
                                         WR_n        : IN  STD_LOGIC;-- inform weather the
operation is a read or a write
                                         GRANT_n     : OUT STD_LOGIC_VECTOR(NumberOfMasters
downto 0));
END MemoryController ;

ARCHITECTURE MC of MemoryController  is
BEGIN
  PROCESS (TPMState, BUSREQ_n, TRANSType,ADDRS)
    VARIABLE address:STD_LOGIC_VECTOR(MemoryAddressSize downto 0):=ADDRS;
    VARIABLE switch : STD_LOGIC_VECTOR(1 downto 0):="11";
    BEGIN
      IF(TRANSType >"01") THEN -- the brust type of operation
        address := (ADDRS + SIZE );
      END IF;
      IF ((address > Memory1Start) AND (address < Memory1End)) THEN
        switch:="00";
      END IF;
      IF ((address > Memory2Start) AND (address < Memory2End)) THEN
        switch:= "01";
      END IF;
      IF ((address > Memory3Start) AND (address < Memory3End)) THEN
        switch:= "10" ;
      END IF;
      CASE switch IS
        WHEN "00" =>
       -- BEGIN
          IF (BUSREQ_n = "10") THEN
            GRANT_n<= "10";
          ELSE
            GRANT_n<= "11";
          END IF;
        --END
        WHEN "01" =>
        --BEGIN
          IF (BUSREQ_n = "10") THEN
            GRANT_n<= "10";
          ELSIF (((BUSREQ_n = "01")AND (TPMState= "01"))) THEN
            GRANT_n <= "01";
          ELSE
            GRANT_n <= "11";
          END IF;
        --END
        WHEN "10" =>
        --BEGIN
          IF (BUSREQ_n = "10") THEN
            GRANT_n<= "10";
          ELSIF (BUSREQ_n = "01" AND TPMState > "01") THEN
            GRANT_n<= "01";
          ELSE
            GRANT_n<= "11";
          END IF;
       -- END
        WHEN OTHERS =>
          GRANT_n<="11";
      END CASE;
END PROCESS;
END MC;
```

RSA engine

```
--
**********************************************************************************************
**
-- RSA_Wrap for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
**********************************************************************************************
**
library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use work.my_library.all;

ENTITY RSA_wrap IS
  PORT (RSA_START   : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RSA_END     : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RSA_DONE    : OUT STD_LOGIC;
        RSA_RSLT    : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RESET_n     : IN STD_LOGIC;
        DATA_ADDRS  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0);
        DATA        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        WREN        : OUT STD_LOGIC;
        RSA_OUT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        RSA_RAM_ADDRS: IN STD_LOGIC_VECTOR (6 DOWNTO 0);-- ADDRESS TO WRITE THE KES ADDRESS
RANGE 0000000 TO 0011111 ADDRESS THE KEYS
        RSA_RAM_DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);--WHILE 0100000 TO 0111111 ADDRESS THE
M VALUES and 1000000 TO 1011111 THE R_C
        RSA_RAM_WREN: IN STD_LOGIC;

        RSA_CLK     : IN STD_LOGIC;
        CLK         : IN STD_LOGIC);
  END ENTITY;

ARCHITECTURE BHV OF RSA_wrap IS
COMPONENT rsa_top is
  port(
    clk       : in  std_logic;
    reset     : in  std_logic;
    valid_in  : in  std_logic;
    start_in  : in  std_logic;
    x         : in  std_logic_vector(15 downto 0);  -- estos 3 son x^y mod m
    y         : in  std_logic_vector(15 downto 0);
    m         : in  std_logic_vector(15 downto 0);
    r_c       : in  std_logic_vector(15 downto 0);  --constante de montgomery r^2 mod m
    s         : out std_logic_vector( 15 downto 0);
    valid_out : out std_logic;
    bit_size  : in  std_logic_vector(15 downto 0)  --tamano bit del exponente y (log2(y))
    );
END COMPONENT;

COMPONENT  RSA_Key_RAM IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                clock           : IN STD_LOGIC  := '1';
                data            : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                wren            : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
```

```
END COMPONENT ;
SIGNAL RSA_RAM_DATA_m: STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL RSA_RAM_ADDRS_m: STD_LOGIC_VECTOR (4 DOWNTO 0);
SIGNAL SW : BIT;
SIGNAL WREN_MOD_m : STD_LOGIC;
SIGNAL WREN_KEY_m : STD_LOGIC;
SIGNAL WREN_R_C_m : STD_LOGIC;
SIGNAL M_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL M_m_i : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL M_m_c : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL Y_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL R_C_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL RESET :STD_LOGIC;
SIGNAL START_IN :STD_LOGIC;
SIGNAL VALID_IN :STD_LOGIC;
SIGNAL VALID_OUT_m: STD_LOGIC;
SIGNAL S_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL RSA_OUT_m: STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL DATA_m :STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL X_m : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL OUT_FLAG :STD_LOGIC;
SIGNAL BIT_SIZE_m :STD_LOGIC_VECTOR (15 DOWNTO 0):= x"0011";
BEGIN
RESET <= NOT RESET_n;
RSA_RAM_ADDRS_GNRT : PROCESS (RSA_CLK, RESET_n)
VARIABLE ADDRS_m : STD_LOGIC_VECTOR(4 DOWNTO 0);
VARIABLE START_IN_m : STD_LOGIC:='0';
VARIABLE START_COUNT : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
BEGIN
  IF (RESET_n = '0') THEN
    ADDRS_m:= (OTHERS =>'0');
    SW<='0';
    START_IN_M:= '0';
    START_COUNT:= "000";
    M_m_i      <= x"b491";
  ELSIF (RSA_CLK'EVENT AND RSA_CLK='1' ) THEN
  IF (START_IN_m='0')THEN -- CONDTION TO START THE RSA ENGINE REQUIRED BY THE DEVELOPER
        IF (START_COUNT = "000")THEN
          START_IN <='1';
          START_IN_m:='0';
        ELSIF (START_COUNT >= "110")THEN
          START_IN_m := '1';
          START_COUNT:="000";
          START_IN <='0';
        ELSE
          START_IN<='0';
          START_IN_m:='0';
        END IF;
        START_COUNT := START_COUNT + "001";
      END IF;-- END OF CONDTION
  IF(START_IN_m = '1' ) THEN
    SW<= NOT SW;
    IF (ADDRS_m = "11111")THEN
      ADDRS_m:= "00000";
    ELSIF (VALID_OUT_m='0')THEN
      ADDRS_m := ADDRS_m + "00001";
    ELSE
      ADDRS_m:= ADDRS_m;
    END IF;
    IF (RSA_RAM_WREN = '1')THEN -- IF THE EE WANT TO LOAD KEYS TO THE RAM
      RSA_RAM_ADDRS_m <= RSA_RAM_ADDRS(4 DOWNTO 0);
      IF (SW = '0') THEN
        RSA_RAM_DATA_m <= RSA_RAM_DATA (15 DOWNTO 0);
      ELSE
        RSA_RAM_DATA_m <= RSA_RAM_DATA (31 DOWNTO 16);
      END IF ;
      IF (RSA_RAM_ADDRS(6 DOWNTO 5)= "01") THEN
        WREN_KEY_m<= '0';
        WREN_MOD_m<='1';
        WREN_R_C_m<='0';
      ELSIF (RSA_RAM_ADDRS(6 DOWNTO 5)= "10") THEN
        WREN_KEY_m<= '0';
        WREN_MOD_m<='0';
        WREN_R_C_m<='1';
      ELSE
        WREN_KEY_m<= '1';
        WREN_MOD_m<='0';
        WREN_R_C_m<='0';
      END IF;
    ELSE     -- KEYS ARE ALREADY LOADDED SO KEEP ON READING

            OUT_FLAG <= '1';
          END IF;
      IF (OUT_FLAG = '1')THEN
      IF (VALID_OUT_m= '0') THEN
        WREN_KEY_m<= '0';
        WREN_MOD_m<='0';
        WREN_R_C_m<='0';
        RSA_RAM_ADDRS_m<= ADDRS_m;
        IF (SW= '0') THEN
          X_m <= DATA_m(15 DOWNTO 0);
        ELSE
          X_m <= DATA_m(31 DOWNTO 16);
        END IF;
      ELSE
        IF(SW = '0') THEN
          RSA_OUT(15 DOWNTO 0) <= S_m;
        ELSE
          RSA_OUT(31 DOWNTO 16) <= S_m;
        END IF;
      END IF;
         END IF;
  END IF;
  END IF;-- END CLK EVENT
END PROCESS RSA_RAM_ADDRS_GNRT;
RAM_MOD :  RSA_Key_RAM PORT MAP(address=>RSA_RAM_ADDRS_m,clock   => RSA_CLK,data =>
RSA_RAM_DATA_m, wren    => WREN_MOD_m,  q => M_m_c      );
RAM_KEY :  RSA_Key_RAM PORT MAP(address=>RSA_RAM_ADDRS_m,clock   => RSA_CLK,data =>
RSA_RAM_DATA_m, wren    => WREN_KEY_m,  q => Y_m);
RAM_R_C :  RSA_Key_RAM PORT MAP(address=>RSA_RAM_ADDRS_m,clock   => RSA_CLK,data =>
RSA_RAM_DATA_m, wren    => WREN_R_C_m,  q => R_C_m      );
RSA     :  rsa_top PORT MAP ( clk => RSA_CLK, reset=> RESET, valid_in => VALID_IN, start_in =>
START_IN,
                               x   =>X_m,y => Y_m, m => M_m,r_c =>R_C_m,s => S_m, valid_out =>
VALID_OUT_m,
                                bit_size => BIT_SIZE_m);
M_m<= M_m_i WHEN RESET_n = '0' ELSE M_m_c;
RSA_DATA_ADDRS_GNRT: PROCESS (CLK, RESET_n)
VARIABLE DATA_ADDRS_m :STD_LOGIC_VECTOR (9 DOWNTO 0);
VARIABLE RTRN_ADDRS_m: STD_LOGIC_VECTOR (9 DOWNTO 0);
VARIABLE OUT_COUNT:STD_LOGIC_VECTOR (4 DOWNTO 0):= "00000";
VARIABLE DONE_m: STD_LOGIC;
VARIABLE VALID_IN_m : STD_LOGIC:='0';
BEGIN
  IF (RESET_n = '0') THEN
      DATA_ADDRS_m := (OTHERS=> '0');
      RTRN_ADDRS_m := (OTHERS => '0');
      VALID_IN_m :='0';
      DONE_m := '0';
    ELSIF (CLK'EVENT AND CLK= '1') THEN
      DATA_ADDRS <= DATA_ADDRS_m;
      RSA_DONE <= DONE_m;
      DATA_m<= DATA;
      RSA_OUT<= RSA_OUT_m;
      IF(VALID_OUT_m ='0') THEN -- NO OUT PUT IS READY SO IT IS OKAY TO READ
      DONE_m := '0';
      OUT_COUNT :="00000";
      IF (DATA_ADDRS_m = "0000000000") THEN
        DATA_ADDRS_m := RSA_START;
        VALID_IN_m :='1';
      ELSIF (DATA_ADDRS_m >"0000000000" AND DATA_ADDRS_m< RSA_END)THEN
        DATA_ADDRS_m := DATA_ADDRS_m + "0000000001";
        VALID_IN_m := '1';
      ELSE
        VALID_IN_m := '0';
      END IF;
      END IF; --END THE CONDTION THAT CHECKS THE WRITE AND READ
      IF (VALID_OUT_m= '1') THEN


      END IF;

      IF (VALID_OUT_m = '1') THEN  -- OUT ADDRESS COUNTING
        WREN    <='1';
        RTRN_ADDRS_m:= DATA_ADDRS_m;
        IF (OUT_COUNT = "11111" )THEN
          DONE_m:='1';
          WREN<= '0';
          DATA_ADDRS_m:= RTRN_ADDRS_m;
       -- END;
        ELSIF (OUT_COUNT = "00000") THEN
          WREN    <='1';
          RTRN_ADDRS_m:= DATA_ADDRS_m;
          DATA_ADDRS_m := RSA_RSLT;
        ELSE
       -- BEGIN
          DATA_ADDRS_m := RSA_RSLT + "0000000001";
        END IF;
        OUT_COUNT := OUT_COUNT +"00001";
      END IF;-- END OUT ADDRESS COUNTING IF

    END IF; ---CLK EVENT AND RESET

END PROCESS RSA_DATA_ADDRS_GNRT;

END BHV;
```

SHA-1 engine

```
--
*********************************************************************************************
**
-- Top TCM for TCM
-- Version 0.1
-- Designed by Mussie Tesfaye
-- Modified 01/08/2012
--
*********************************************************************************************
**

library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use work.my_library.all;

ENTITY SHA1_wrap IS
  PORT (SHA_START   : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_END     : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_DONE    : OUT STD_LOGIC;
        SHA_RSLT    : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
        RESET_n     : IN STD_LOGIC;
        DATA_ADDRS  : OUT STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0);
        DATA        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        WREN        : OUT STD_LOGIC;
        --OUT_ADDRS   : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
        SHA_OUT     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        CLK         : IN STD_LOGIC);
  END ENTITY;

ARCHITECTURE BHV OF SHA1_wrap IS
COMPONENT sha1 is
  port(
    m                : in  bit_vector ( 31 downto 0); -- 32 bit data path require 16 clock to
load all 512 bits of each block
    init             : in  bit;                       --    initial message
    ld               : in  bit;                       --    load signal
    h                : out bit_vector ( 31 downto 0); --    5 clock after active valid signal
is the message hash result
    v                : out bit;                       --    hash output valid signal one clock
advance
```

```
    clk              : in  bit;                        --    master clock signal
    rst              : in  bit                         --    master reset signal
  );
END COMPONENT;
SIGNAL OUT_m : BIT_VECTOR (31 DOWNTO 0);
SIGNAL DATA_m : BIT_VECTOR (31 DOWNTO 0);
SIGNAL LD_m :BIT;
SIGNAL INIT_m : BIT;
SIGNAL DONE_m: STD_LOGIC;
SIGNAL RST_m :BIT;
SIGNAL V_m :BIT;
SIGNAL CLK_m:BIT;
BEGIN
  CLK_m<= To_Bit(CLK);
PROCESS (CLK, RESET_n)
  VARIABLE DATA_ADDRS_m :STD_LOGIC_VECTOR (9 DOWNTO 0);
  VARIABLE RTRN_ADDRS_m: STD_LOGIC_VECTOR (9 DOWNTO 0);
  VARIABLE OUT_FLAG: STD_LOGIC;
  VARIABLE OUT_COUNT:INTEGER RANGE 0 TO 10;
  BEGIN
    IF (RESET_n = '0')THEN
   -- BEGIN
      DATA_ADDRS_m := (OTHERS=> '0');
      RTRN_ADDRS_m := (OTHERS => '0');
      INIT_m <= '1';
      RST_m <= '1';
      WREN <='0';
      DONE_m<= '0';
   -- END ;-- END RESET BEGIN
    ELSIF (CLK'EVENT AND CLK= '1') THEN
   -- BEGIN
      RST_m <='0';
      DATA_ADDRS <= DATA_ADDRS_m;
      --OUT_ADDRS <= OUT_ADDRS_m;
      SHA_DONE <= DONE_m;
      DATA_m<= To_BitVector(DATA);
      SHA_OUT<= To_StdLogicVector(OUT_m);
      IF(OUT_FLAG ='0') THEN -- NO OUT PUT IS READY SO IT IS OKAY TO READ
      IF (DATA_ADDRS_m = "0000000000") THEN
     -- BEGIN
        DATA_ADDRS_m := SHA_START;
        LD_m<='1';
     -- END;
      ELSIF (DATA_ADDRS_m >"0000000000" AND DATA_ADDRS_m< SHA_END)THEN
      --BEGIN
        DATA_ADDRS_m := DATA_ADDRS_m + "0000000001";
        LD_m<='1';
        INIT_m <='0';
     -- END;
      ELSE
        LD_m<='0';
      END IF;
      END IF; --END THE CONDTION THAT CHECKS THE WRITE AND READ
      IF (V_m= '1') THEN
        OUT_COUNT :=0;
        OUT_FLAG:='1';
        WREN    <='1';
        RTRN_ADDRS_m:= DATA_ADDRS_m;
      END IF;
IF (OUT_FLAG = '1') THEN  -- OUT ADDRESS COUNTING
     -- BEGIN
        IF (OUT_COUNT = 4 )THEN
        --BEGIN
          OUT_FLAG:='0';
          DONE_m<='1';
          WREN<= '0';
          DATA_ADDRS_m:= RTRN_ADDRS_m;
       -- END;
        ELSIF (OUT_COUNT = 0) THEN
          DATA_ADDRS_m := SHA_RSLT;
        ELSE
       -- BEGIN
          DATA_ADDRS_m := SHA_RSLT + "0000000001";
        END IF;
        OUT_COUNT := OUT_COUNT +1;
      END IF;-- END OUT ADDRESS COUNTING IF
    END IF; ---CLK EVENT AND RESET
    END PROCESS;
    SHA1_U1 : sha1 PORT MAP ( DATA_m, INIT_m , LD_m, OUT_m, V_m, CLK_m, RST_m );
END BHV;
```

My_Library

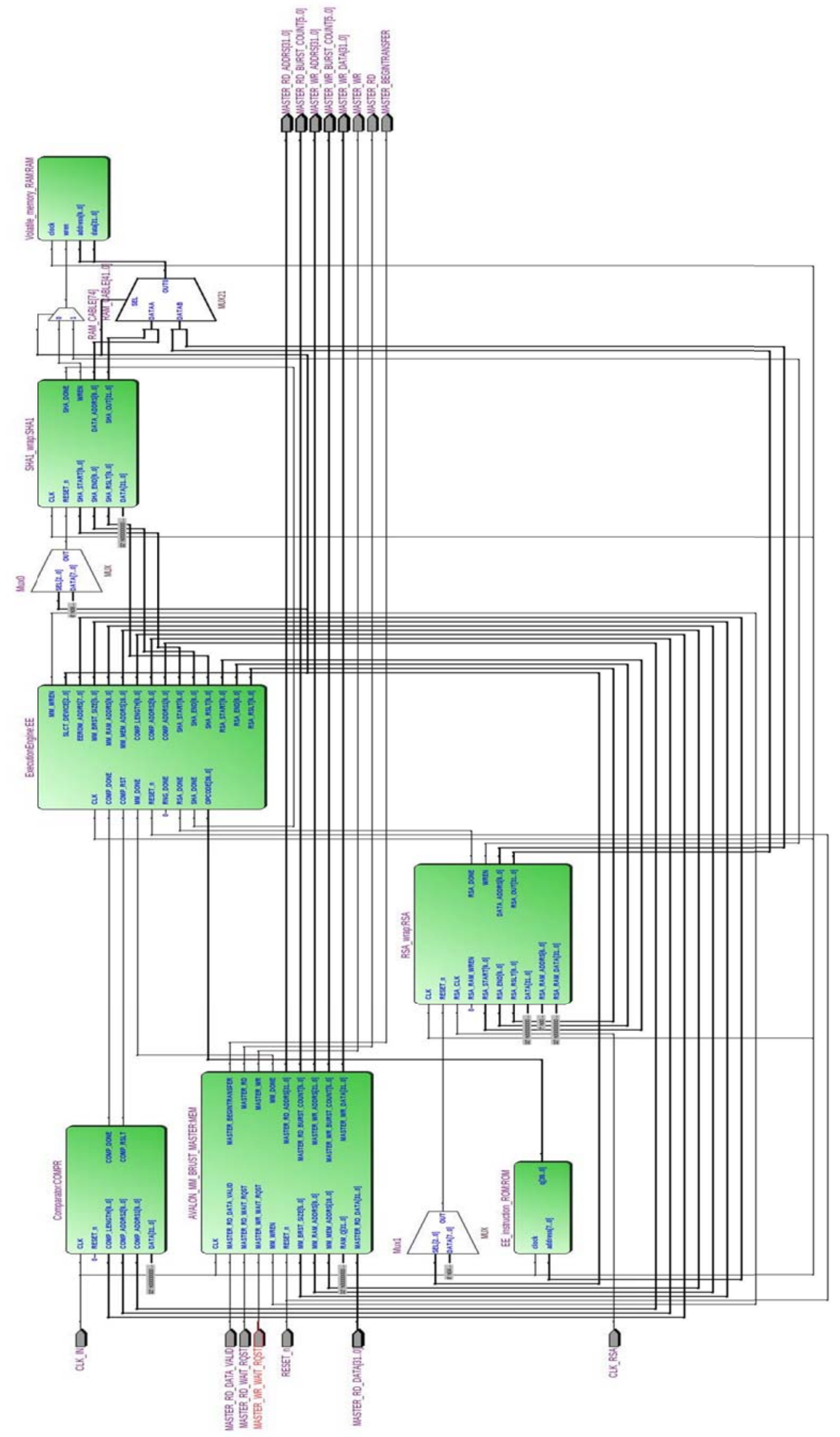
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
package my_library is
-- collection of types and components to be used through out the project
TYPE ARY IS ARRAY  (0 TO 3 ) OF STD_LOGIC_VECTOR (31 DOWNTO 0);
SUBTYPE int IS INTEGER RANGE -128 TO 127; -- might consider range -512 to 512
constant MemoryAddressSize: int :=9;-- for memory address of width of 10 bit RAM
CONSTANT NumberOfMasters  : int :=1; --represent two manster one TCM and the other is the CPU
CONSTANT DATA_WIDTH       : int :=7; -- FOR AN 8 BIT DATA RANGING FROM 7 DOWNTO 0
PROCEDURE MEMORY_RW (SIGNAL TRANSType       : IN  STD_LOGIC_VECTOR (1 downto 0);-- type of
transfer weather it is 00-IDEL; 01-BUSY; 10-NONSEQ; 11-SEQ
                    SIGNAL   SIZE                     : IN  STD_LOGIC_VECTOR(2 downto 0);--
size of the transfer need to be checked for brust operation
                    SIGNAL   GRANT_n     : IN STD_LOGIC_VECTOR(NumberOfMasters downto 0);
                    SIGNAL   CLK         : IN STD_LOGIC;
                    VARIABLE DONE        : OUT STD_LOGIC;
                    VARIABLE ADDRS_r     : INOUT STD_LOGIC_VECTOR (MemoryAddressSize downto
0);
                    VARIABLE ADDRS_w     : INOUT STD_LOGIC_VECTOR (MemoryAddressSize downto
0));
end my_library;

PACKAGE BODY my_library IS

  PROCEDURE MEMORY_RW (SIGNAL TRANSType     : IN  STD_LOGIC_VECTOR (1 downto 0);-- type of
transfer weather it is 00-IDEL; 01-BUSY; 10-NONSEQ; 11-SEQ
                      SIGNAL   SIZE                   : IN  STD_LOGIC_VECTOR(2 downto 0);--
size of the transfer need to be checked for brust operation
                      SIGNAL   GRANT_n     : IN STD_LOGIC_VECTOR(NumberOfMasters downto 0);
                      SIGNAL   CLK         : IN STD_LOGIC;
                      VARIABLE DONE        : OUT STD_LOGIC;
                      VARIABLE ADDRS_r     : INOUT STD_LOGIC_VECTOR (MemoryAddressSize downto
0);
                      VARIABLE ADDRS_w     : INOUT STD_LOGIC_VECTOR (MemoryAddressSize dOWNTO
0)) IS
  VARIABLE TEMP_r, TEMP_w : STD_LOGIC_VECTOR (MemoryAddressSize DOWNTO 0);
  VARIABLE i:STD_LOGIC_VECTOR (2 DOWNTO 0);
  BEGIN
    i:=SIZE;
   -- WHILE i>"000" LOOP
    IF (GRANT_n = "00"  AND CLK = '1') THEN
        IF (TRANSType ="11") THEN
          IF (i/= "000") THEN --FOR i IN 0 TO CONV_INTEGER (UNSIGNED( SIZE)) LOOP
            TEMP_r:=ADDRS_r;
            TEMP_w:=ADDRS_w;
            ADDRS_r := TEMP_r + "0000000001";
            ADDRS_w := TEMP_w + "0000000001";
            i:= i-"001";
          ELSE
            DONE := '1';
          END IF; -- FOR LOOP
        ELSE
              DONE := '1';
              ADDRS_r := ADDRS_r;
              ADDRS_w := ADDRS_w;
        END IF; -- END SEQUENTIAL TRANSFER
    END IF; -- ACCESS NOT GRANTED
   -- END LOOP;
  END MEMORY_RW;
END  my_library;
```

Appendix B. Simulation results

TCM Netlist



TRITA-ICT-EX-2012:229