DESIGN OF SECURE BOOT PROCESS FOR RECONFIGURABLE ARCHITECTURES

by

Ali Shuja Siddiqui

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2020

Approved by:

_____
Dr. Fareena Saqib

_____
Dr. Arindam Mukherjee

_____
Dr. Madhav Manjrekar

_____
Dr. Milind V. Khire

ABSTRACT

ALI SHUJA SIDDIQUI. Design of Secure Boot Process for Reconfigurable Architectures. (Under the direction of DR. FAREENA SAQIB)

VLSI advancements have enabled proliferation in the Internet of Things(IoT) domain where small scale System-on-a-Chip (SoC) are employed as sensors or actuators. IoT devices are connected with other IoTs and with backend facilities. In today's world IoT is ubiquitous and pervasive, as system designers continue to use IoT based designs. In order to improve the lifespan of a device, hardware reconfigurability has security and functional applications for IoTs. Connectivity, while essential to the operation for an IoT device, also acts as a door for malicious actors. This work explores and identifies the threats to emerging IoT in the spaces of authentication, integrity, confidentiality and communication. For reconfigurable devices, the work extends boot time security and provides solution for Over the Air update mechanisms for reconfigurable architectures. This work uses automotive and smart grid to demonstrate applications of the research outcomes.

DEDICATION

I would to like to begin by thanking God Almighty for giving me the opportunity to travel half way around the world to earn my education, for providing for me and specially for giving me ideas when nothing seemed to work.

I dedicate this work to my parents and my siblings, who have always believed in me and have encouraged me to further myself. I thank them for all the support, wisdom and the strength they have given me and for being with me when I needed them the most.

This work is also dedicated to my wife Yameena, for being supportive, patient and encouraging throughout. Her support made this degree a lot easier.

Lastly, I dedicate this work to my friends, Yutian Gui, Manikanta Bhagwatula, Harsha Ganti, Amit Singh and Suyash Mohan Tamore.

ACKNOWLEDGEMENTS

I would like to thank and acknowledge the committee members for being a part of my committee and for putting in the time and effort to make my research better. I would also like to acknowledge the Graduate School for providing me with GASP grants that helped me tremendously towards my degree. This research has been funded by National Science Foundation Grant "CRII: SaTC: Hardware based Authentication and Trusted Platform Module functions (HAT) for IoTs", NSF Award Number 1819687.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Motivation

There is an estimated 19.4 billion connected devices in the year 2019, and this number is expected to increase to 34.2 billion by the year 2025 [9]. It consists of devices connected wirelessly, via a wired connection or through some proprietary network stacks. The connection may either be one directional or bi-directional.

Networked devices have pervaded every domain of life in many forms. In the form of computers, laptops tablets, mobile phones, smart watches, and other small scale devices. Depending on the application, these devices may exist as smart appliances, industrial sensors and actuators, and personal smart devices. They may be connected with a comparatively resourceful backend server with or with an array of similar devices in a mesh or similar network. Small scale devices are resource constrained in terms of computing and storage. These limitations impact the effectiveness of the device in the field. This requires systems designers to carefully plan the resources that are available on a device, and plan what features can be effectively implemented on the device.

When a system is designed, the basic two components of design are firstly the hardware and then the software stack. The software stack is only limited by the hardware limitations. In terms of fixed fabricated hardware, once the hardware has been fabricated, it cannot be altered in any way. In case there is some fault later discovered in the hardware or an improvement in the architecture, it cannot be integrated. This limitation can be overcome by integrating reconfigurable hardware design. Reconfigurable hardware, such as FPGAs have an array of hardware components implemented on the fabric of a device. This fabric can be configured after the device fabrication

using in-field reconfigurable architectures.The use of FPGAs in the market is only expected to increase from $63 billion in the year 2019 to $117.97 billion by the year 2026[10] [11].

Connected devices open a device to unwanted, rogue and malicious actors. These actors pose various threats to the operations of the connected devices. This may lead to sensitive information leakage, communication corruption and even malicious modification of the on-board software and in the case of reconfigurable devices, hardware configuration. This research focuses on addressing issues in resource constrained reconfigurable hardware and presents novel hardware-based approaches.

1.2 Contributions

This research has following contributions towards improving boot and runtime security in reconfigurable architectures:

- Identifies security threats and presents threat models that affect boot and runtime security of an IoT in the field. Existing work and their shortcomings are also discussed.

- Proposes a novel boot time security and Over-the-Air update framework for FPGAs. This scheme employs hardware assisted cryptographic systems, such as Trusted Platform Module (TPM) to present a key update and a secure communication scheme. To implement secure over-the-air update mechanism, the server verifies the integrity of the client device. This ensures in-field security compliance.

- Presents a novel mechanism for implementing FPGA based logic obfuscation. Pre-boot in-field device authentication scheme is extended to implement runtime logic obfuscation. This obfuscation method is to best of knowledge the first of its kind. This scheme manipulates application bitstream before being deployed onto a device. Once the device has been authenticated with the con-

tent provider, only then will it provide the device with correct and complete bitstream configuration. In combination with the logic obfuscation, this scheme uses key based logic locking to improve logic runtime bitstream security.

- Proposes hardware based secure communication and secure code isolation framework for vehicular Electronic Control Units (ECUs).

- Provides a key provisioning scheme for IoT devices in the power grid and puts forward a design for secure reconfigurable power converters for the power grid.

1.3 Organization

This document is divided into the following chapters. Chapter 2 describes the background information on the topics involved in this research. It also gives an overview on the history and the progression of the existing work performed in the research and commercial spheres. The remaining chapters explain the research conducted.

Chapter 3 details the contributions on proposing secure boot and over-the- air update mechanisms for securely updating FPGA bitstreams. Chapter 4 presents a mechanism for implementing runtime logic locking and bitstream logic obfuscation scheme. Chapter 5 extends the research to secure communication and secure code execution in the automotive domain. Chapter 6 proposes key provisioning and in the smart-grid and a design for secure reconfigurable power converters using FPGAs. Lastly, conclusions of the research are presented in Chapter 7.

CHAPTER 2: Background Study on Secure Boot and Overview of Hardware Security Primitives

In this capter, we will overview the security concepts and hardware security primitives for system integration.

2.1 Internet of Things

The term Internet of Things (IoT) was first coined in the year 1999 by Kevin Ashton[12]. Initially, Internet of Things was designed for the purposes of supply chain management, but with time, its application scope was widened to include other domains as well. IoT is the network of connected devices which are automated in nature and are used for sensing and/ or actuation. Sensing based IoTs result in generating knowledge base for taking decisions on, whereas an IoT can use different actuation mechanisms to manipulate with the environment it is located in.

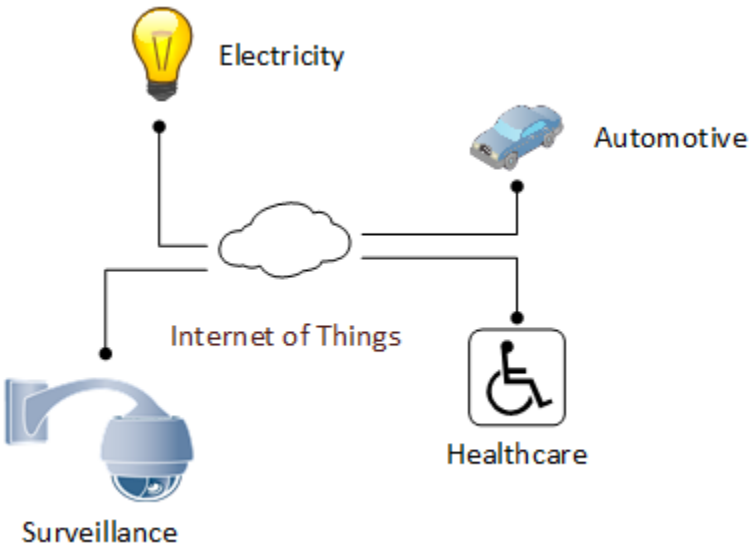

Figure 2.1: Common Applications of Internet of Things.

IoT systems consists of three entities, the backend, the middleware, and the end

device. Depending on the scale and application of the system, the backend can be a personal computer, a dedicated server to even a scalable cloud environment. For sensing applications, where an IoT device is resource constrained and cannot make decisions itself, the decision-making is left at the backend. The information gathered from sensors is sent off to the backend, where it can take an informed decision based on the information gathered. Big Data analytics has gained traction in IoTs for data handling[13]. The middleware consists of the connecting technologies that connect an end device with the backend. It comprises not just the networking elements, such and router and switches etc., but also of any intermediary data collection and decision-making nodes that lie between the backend and the end device[14].

For the end devices in an IoT system, there are various technological factors that have resulted in the rapid adoption and growth of IoTs over the years. Two of such factors are the constant improvement in the embedded systems computing domain and the wireless connection mediums such as Wi-Fi, Bluetooth Low Energy (BLE) and Zigbee, etc., With each iteration of these technologies, the focus has been on improving speed of the network, the cost of computing and power efficiency[15].

Internet of Things have found its ground in several areas, including consumer electronics, smart grid, automotive industry and automation. For the purpose of this research, I have focused my efforts on IoT devices in the smart grid.

2.1.1 Smart Grid

With the introduction of smart grid, the energy grid is no longer comprised from the uni-directional (i.e. from the backend to the end nodes) data communication path, but bi-directional. All electrical and computation-oriented devices generate data that is communicated over a data communication network to the backend Supervisory Control and Data Acquisition system (SCADA). Additionally, end consumers also participate on the network using IoT devices, e.g. smart meters, smart appliances and even electrical car chargers which exchange data with the smart grid middleware.

Smart grid allows inclusion of Distributed Energy Resources (DER) to the grid. DERs are power generation systems that contribute to the electrical grid by producing power. The generated power can come from different sources such natural gas, hydro, wind and solar etc. DER are equipped with IoT devices that communicate with the SCADA backend. The communication between DER and SCADA is performed using the standard IEC 61850[16].

2.1.2 Automotive Security

Vehicles are composed of individual systems that are connected together. They are mechanical and electronic in nature. These systems include engine, steering[17], brakes, air conditioning, the infotainment system, parking assistance, etc. Modern vehicles are pushing towards increasingly making interaction between all on-board systems digital. As such, instead of using the physical state of a system an Electronic Control Unit (ECU) is connected, which broadcasts the state of the physical system as a data message. All ECUs are connected with each other over a network. Currently, there are several network protocols being used. Some of the more widely used are Controller Area Network (CAN), CAN with Flexible Data-rate (CANFD), and EtherCAT etc.

ECUs had originally started as a way to fine tune an engine on a vehicle. However, in 1996, it was made mandatory that all vehicles provide access to On-Board Diagnostics using an OBD-II port. This law pushed car manufacturers to add a digital interface to the components embedded in a car. Nowadays, automation in automotive has progressed enough to offer self-driving vehicles. Additionally, cars are also connecting with each other and with a growing roadside infrastructure to form a V2V and V2X infrastructure[18].

2.2 Reconfigurable Architectures

Application Specific Integrated Circuits (ASICs) are fabricated circuits that designed to perform one task. Whereas microprocessors, on the other hand provide freedom of what tasks can be performed on it. Tasks that can be performed on a microprocessor are dictated by code that is executed on it. The ability of a microprocessor is defined by its Instruction Set Architecture (ISA). Code itself cannot modify the underlying architecture.

Reconfigurable architectures, most popular being the Field Programmable Gate Arrays (FGPAs) lie in between the spectrum of a microprocessor and an ASIC. FPGA allow reconfiguration of hardware. FPGAs are composed of Programmable Logic fabric (PL) consisting of programmable logic gates that are connected together to form a digital logic circuit. The gates in an FPGA are made of Look-up Tables (LUTs). Current generation of Xilinx FPGAs are composed of either five input or 6 input LUTs. Additionally, based on the size, and application target, FPGAs also provide adders, registers, block memory, Digital Signal Processing (DSP) components, as well as embedded microprocessors(PS). Following are the introductory FPGA concepts that are referenced and used in this research:

- **Slice**: Slice is a collection of logic components in an FPGA. In Xilinx's 7000 series of FPGAs, each slice consists of four LUTs, eight storage elements (flip-flops and latches), carry logic elements and multiplexers. There are two different kinds of slices, namely SLICEM and SLICEL. A SLICEL can only contain the components listed above, whereas SLICEM can contain additional components such as distributed RAM components as well as 32-bit shift registers.

- **Configuration Logic Block (CLB)**: CLBs consists of two slices. These slices can either be a combination of SLICEL and SLICEM or two SLICELs.

- **PCAP and ICAP**: PCAP stands for Processor Configuration Access Port

and ICAP stands for Internal Configuration Access Port. An ICAP port is a microprocessor based (PS) interface to the PL. PCAP interface provides capabilities for reading and writing CLBs LUTs and memory elements as well partial bitstream loading and bitstream readback. At the time the FPGA boots up, the access to the PL fabric is first given to PCAP. FPGAs also provide an additional Internal Configuration Access Port (ICAP). This port is used by PL fabric internally to modify itself on the run.

- **Dynamic Partial Reconfiguration**: When an FPGA needs to be reprogrammed, the execution of the PL must first be shutdown, the updated bitstream is copied to the fabric and then the fabric can be brought up again. Dynamic Partial Reconfiguration mitigates the issue of shutdown by enabling allowing areas of the fabric to be set as Dynamic Partially Reconfigurable areas. Such areas can be reprogrammed on the fly. The surrounding logic around these areas stays the same throughout the operation. One common use case of DPR is to have an area that can be used to implement multiple implementations of the same design entity.

2.3 Threat Models

There is plethora of vulnerabilities found in IoT devices ranging from remote network-based vulnerabilities, local as well as physical vulnerabilities. Different components of this research delve into presenting solutions against the following threats faced at different levels of operation of a device.

2.3.1 Eavesdropping, Replay and Man in the Middle attacks

In a network of devices, all devices with access to a medium can access information travelling over that medium. As such, there is always a chance for an eavesdropper listening to the conversation occurring over the medium. If the communication is not encrypted, the eavesdropper can listen to this conversation and extract sensitive

information. Additionally, this information can be replayed to introduce unexpected behavior at an unsuspecting victim. Furthermore, a malicious actor can then perform a man in the middle attack by posing as a trusted node and communicating with a legitimate node on the network.

An example of this attack is the 2015 attack that was performed on Jeep Cherokee[19]. The vehicle uses a service called Uconnect which runs on the car's infotainment dashboard. This service connects to the internet to communicate with a backend infrastructure. The hackers got access remote access to the infotainment system using a remote vulnerability found on the service. The infotainment system is connected with the rest of the car ECUs using CAN bus. Hackers were able to communicate with the other systems using CAN messages. They were able to have total control over systems such as braking, steering, AC etc. Since the ECUs connected over the CAN network perform no identity check, they accepted and reacted to all the malicious information being broadcasted.

2.3.2 Private Key Exposure

For implementing private communication between two nodes over an untrusted network, encryption is used. Encryption uses one-way function and an encryption key to convert plain text input into cipher text. In case of symmetric key encryption e.g. AES encryption, the encryption key is shared between the two communicating parties beforehand. In case this key is leaked to an eavesdropping adversary, the adversary can decrypt the entire flow of traffic between the nodes sharing the key. The effect is exacerbated when a collection of devices uses the same key. Therefore, it is a priority to store private keys in a secure area on a device. Recently, hackers were able to retrieve encryption keys to a database encrypting personal information on staying guests at a hotel chain[20].

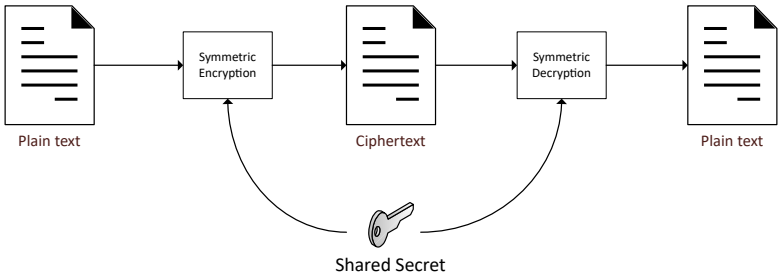

Figure 2.2: Symmetric Encryption

In case of asymmetric cryptography, for each node there is a pair of keys generated, a pair of public and private keys. The public key can be distributed whereas the private key must be key in a safe environment. A node uses its private key to sign a message. When the message is received at the recipient, the public key of the sender is used to verify the signature received. If the private key of a node is compromised, a malicious node can pose as the sender.

2.3.3 Unauthorized Modification to Firmware and Reconfigurable Logic Configuration

A microprocessor-based system performs the actions that are defined by the code existing on its memory. The code may be low level firmware code, operating system or any application. Depending on the system, it can also be bare metal application that is application centric and does not require an additional operating system layer. The interest of an attacker is to redirect the normal flow of execution to an unauthorized piece of code[21][22].

In reconfigurable computing domain, SRAM based FPGAs allow modification in the field. In SRAM based FPGAs, PL is populated at boot time. This process is either performed by the Zeroth Stage Boot Loader software commonly referred to as BootROM[23] or by the First Stage Boot Loader (FSBL)[24] depending on the type of FPGA. If the FPGA is equipped with a PS, it is the responsibility to load the PL bitstream, otherwise BootROM takes care of the PL bitstream loading process.

A bitstream can also be modified at runtime if the target FPGA is equipped with a PCAP or ICAP port. In an FPGA, an attacker is interested in modifying the bitstream. An attacker can either replace the PL logic to perform entirely different tasks, add or remove functionality (e.g. hardware trojan), or may even add a leakage side channel for secret information extraction [25]. There are two points of attack for an attacker namely, at boot or during runtime. At boot, before the bitstream is loaded, an attacker may replace the bitstream with a malicious bitstream. Whereas at runtime, once the bitstream has been loaded an attacker may target dynamic reconfigurable partitions or may want to target certain portions of the configuration. To achieve this, an attacker can use the PCAP or the ICAP.

2.3.4 Nonsecure Communication with Content Provider

For an FPGA device placed in the field, bitstream updates can be provided manually physically by an engineer, through a physical update mechanism or through the use of remote updates over a network. If a content provider over a network is not secure, an adversary may spoof its identity to become a content provider. Therefore, an adversary may be able to push malicious updates to the client. On the other hand, an adversary can also impersonate a device on the field to download bitstream updates from a content provider not meant for it.

2.4 Security Concepts

2.4.1 Confidentiality, Integrity and Availability

Confidentiality, Integrity and Availability (CIA)are core concepts of information security. This triad is a notion of guiding security policies. Confidentiality refers to the property that no unauthorized party is able access to secure information in a system. Confidentiality can be achieved through two ways. One way is by guaranteeing that communicating parties are physically in an isolated environment. The other way is to encrypt communication between the two communicating parties. Encryption allows

communicating parties to exist in an unsecure environment while freely communicating with each other. Integrity is a guarantee that all information part of a process is safe from any unauthorized modification from a malicious actor or from a compromised authorized actor. Integrity can be ascertained through various scheme. Once common scheme is by the use of MAC or Message Authentication Code. (MAC) or Key Hashed Message Authentication Code (HMAC). Availability ensures that even under less favorable conditions the system is still operational. This can be achieved using redundancy and isolation.

2.4.2 Physical Unclonable Functions

Physical Unclonable Functions (PUF) are a new type of cryptographic primitive used in hardware security to implement identity or secret keys. They rely on the inherent manufacturing process variations, which are used to produce reliable and device-unique output [26][6]. PUFs are based on a challenge-response pair (CRP) mechanism. Challenges are input to the PUF circuit. They are defined using a string of 0s and 1s. The length of the challenge input is decided by the PUF implementation. PUF output, or its response is produced by the combination of the challenge input and the fabric variation caused due to the challenge input.

There two different types of PUFs, namely Weak and Strong PUFs. A major difference between the two is that the weak PUFs have few challenges for which they can uniquely generate a key whereas strong PUFs have a large challenge space and therefore have a unique response for most of the challenges. The bit generation for cryptographic applications is a two-step process that is enrollment and registration. During the enrollment process, each PUF is given a set of challenges and the response pairs are recorded. Later when the PUF is in field, these responses are regenerated for use in identification and encryption applications.

2.4.3 Secure Attestation

In the current landscape with IoTs, connectivity of a device dictates its reach. On the other hand, connectivity also makes a device vulnerable to different security issues. There is a need to maintain integrity of the state of the device. To be certain that the device has not deviated the concept of secure attestation has been introduced. Secure attestation introduces the concept of a prover and a verifier. These two entities work together with each other to verify the integrity of a system. A verifier is a trusted party on the system. The prover on the other hand cannot be trusted initially and must earn its trust with the verifier. Verifier's task is to query the prover with the set of challenges. The prover in turn responds to these challenges by collecting data from the running system. Verifier based on the responses can attest the integrity of the system.

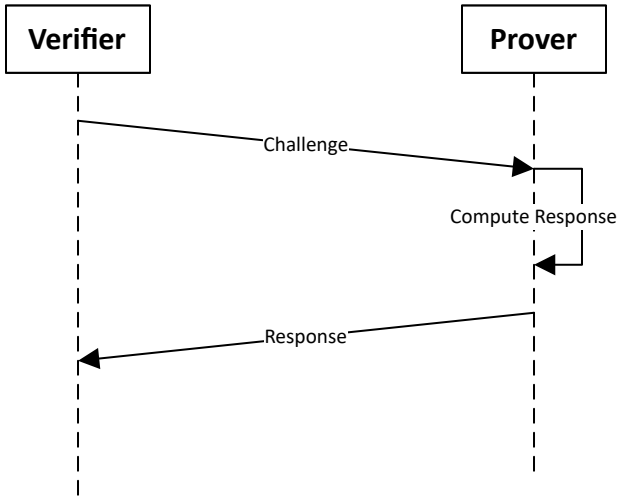

Figure 2.3: Secure Attestation in devices.

There has much work on software attestation in embedded devices. SWATT[27] is one of the earlier works in this domain. This work implements a software attestation framework on over-the-shelf components. It uses time as a measure for integrity. The verifier keeps a model for timing for response times. In case there is deviation, it

corresponds to an attacker who has compromised the system to use code redirection to get to the prover function. The verifier the time difference with the reference value and may deny attestation.

2.4.4 Trusted Platform Module

Trusted Platform Module (TPM) is a hardware module that implements cryptographic functions. These functions can be encryption, data signing and data sealing. TPMs follow TPM specifications put forward by the Trusted Computing Group (TCG)[28]. All TPM implementation must follow the specifications however, the specifications do provide flexibility in terms of the functionality it can provide. TPM also has a limited tamper resistant non-volatile memory. This memory can be used for storing cryptographic objects including keys and other user defined values. There are currently two specifications that are being followed are TPM 1.2 and 2.0. However, there is a shift from TPM 1.2 to TPM 2.0 due to the advanced features that TPM 2.0 provides. An overview of the differences between the two standards is given in Table 2.1.

Table 2.1: Comparison between TPM 1.2 and TPM 2.0

Algorithm	RSA 1024	RSA 2048	ECC NISTP256	ECC BNP256	AES 128	AES 256	SHA-1	SHA-2
TPM 1.2	Yes	Yes	No	No	Optional	Optional	Yes	No
TPM 2.0	Optional	Yes	Yes	Yes	Yes	Optional	Yes	Yes

2.4.5 Secure Boot

Secure Boot is a secure attestation mechanism to establish Root of Trust at boot time. This mechanism works by attesting each layer of software before it can be executed on a system. Since the first layer of execution is the firmware or BootROM, trust is first established at this level. Each next level is first attested by the running layer before execution can be passed to it. As such, in a typical system, where succeeding the layers of execution are firmware, operating system, user applications,

etc., the firmware will attest the operating system, the operating system will attest the user applications and so on.

Secure boot is also commercially available as a part of personal computers. Commercial secure boot implementations (e.g. Microsoft Windows, Trusted Grub, UEFI etc.) rely on TPM to provide a trust anchor. TPM offers provisions for implementing measurable boot using Platform Configuration Registers (PCR). PCRs are registers that hold cumulative hash values. These registers are populated using TPM_PCR_Extend and or the data streaming enabled TPM_HASH structures. 256 bits of data is hashed using either SHA-1 or SHA-2 hashing algorithm on the TPM. Once the process is completed, the computed SHA value is added to an existing selected PCR value. Equation 1 shows the process of PCR extension.

The boot process can be divided into stages, e.g. firmware, operating system, applications etc. For measuring the boot process, the PCR is computed and verified for the next layer in the process before the execution can be passed to that layer. At the end of the process, the value of the PCR provides sequential attestation of the all the components in the chain.

2.4.6 Secure Over-the-Air Updates (OTA)

Over the Air (OTA) updates is a mechanism for a system to push updates to embedded systems, mobile market, automotive industry and IoT devices. IoT devices placed in a field communicate with a specified backend environment over their lifetime for receiving updates. In such a scenario, an attacker would impersonate the update server to be able to push malicious updates to an IoT device. In case of reconfigurable hardware, this update can be to the reconfigurable logic fabric aboard a device. On the other hand, an attacker may even want to appear as a connected device so that it may download software not meant for it to access. As such there is a requirement for establishment of trust between the two actors.

Secure OTA ensures that an update server and a connected client can trust each

other. One way of establishing this trust is by using identity. Identity can be established using concepts such as digital certification. One the field, during connection establishment, the client and the server verify the communicating node's identity using these certificates. Additionally, to mitigate eavesdropping, encryption is implemented between the two nodes.

2.4.7 Trusted Execution Environment and the ARM TrustZone

Trusted Execution Environment (TEE) provides on-chip environment for trusted code execution. Being on-board makes it difficult for snooping or man-in-the-middle attacks. The environment depending on its implementation provides an isolated cell for running trusted code. TEE gives developers the access to write their own functions, for example, proprietary encryption algorithms. Examples of commercial Trusted Execution Environments are ARM TrustZone, Intel SGX and Intel TXT.

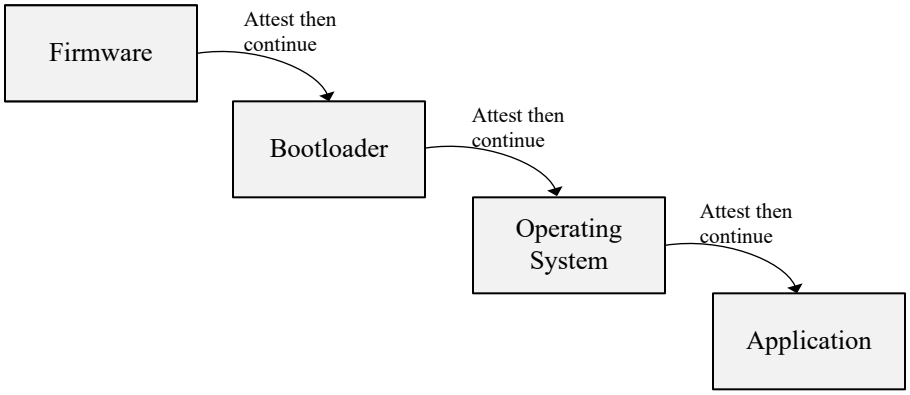

Figure 2.4: Secure Boot Software Chain of Trust.

ARM TrustZone is a Trusted Execution Environment which is implemented within the processor fabric as a co-processor. ARM TrustZone provides isolation for main memory elements, peripherals and provides isolation for system bus elements. It divides the system into two worlds, namely secure and non-secure. The secure world has complete access over all the resources; however, the non-secure world is a configured as an isolated sandbox. The non-secure can be configured to execute general purpose code such as Linux operating system. Secure world can instead run a custom min-

imal operating system that may run some security specific code. Some examples of commercial systems ARM TrustZone are Samsung KNOX, Samsung Pay and Apple TouchID.

2.5 Secure Boot

2.5.1 Early implementations

There is no set singular implementation of secure boot and is dependent on the domain and the security requirements of the system. For the desktop systems, one early approach was the adaption of minimal lightweight kernel design[29]. The kernel was suggested to be minimal and verifiable by manual code inspection. This design forced exokernel design, where every other module excluding the defined core kernel exists in the user space. The purpose of the exokernel is to allocate and dis-allocate memory securely. This scheme pushes the burden of security on the programmer alone by suggesting use of using secure programming practices, inline cross domain function calls and by use of type safe languages as the only form of security. Presently, this scheme has limited practical applicability because of the large code base and short development time.

A microprocessor-based system depends on external code sequence. Basic Input/ Output System (BIOS) historically is the first software code that is executed by a processor. The BIOS performs critical hardware initialization and passes control to a bootloader that loads the operating system. Authors in [30] use BIOS of the motherboard as the root of trust. This secure bootstrap structure is called AEGIS. The software is divided into layers. BIOS holds public keys and digital signatures to verify the integrity of next layer. The architecture also holds recovery alternate code that is loaded if the verification process fails. This system implements a chain of trust. The secure BIOS after verifying the integrity of the bootloader stage using cryptographic hash verification, gives the control to it. The bootloader will subsequently verify the operating system and finally the operating system verifies the software layer. Public

keys are stored in X.509[31] certificates. Shortcoming of such a system is the reliance on external components to implement the security scheme.

[32] is one of the earlier works in integrity checking for intrusion detection. Integrity plays an integral role in measurable boot. This paper presents a software system written checking integrity in a Unix environment. The scheme keeps track of new files, modifications and deleted files during execution. This is useful in detecting an attacker making unexpected changes to the running root file system. The system stores difference with the original files and provided support for MD5 signature for maintaining the integrity of files. However, the attack to the file system can be extended to the tripwire system such that the attacker can cover up their tracks.

2.5.2 Secure Boot Solutions in Desktops

Unified Extensible Firmware Interface (UEFI) is a modern alternative and a replacement to BIOS in desktop PCs. It offers several boot time and runtime services, including support for device drivers, an operating system (OS) independent time service, runtime variable store, UEFI booting and secure boot. To mitigate the possibility of executing unauthorized malicious bootkits[33], UEFI based secure boot enables execution of digitally signed OS loaders that exist in UEFI executable formats (EFI). Approved OS loaders are signed against the UEFI's master signatures. These signatures are verified every time the OS loader is executed. UEFI based secure boot support does not extend to the operating system. Every operating system has their own version of measured boot.

Of the current commercial proprietary and open source operating systems, Microsoft Windows has one of the more popular implementations of measured boot. It uses TPM to support measured boot. At the end of secure boot, UEFI sends the hash of the next software components, i.e. the kernel, Early Launch Anti-Malware (ELAM) drivers and boot drivers, to the TPM. Additionally, TPM is also used for signing the collected measured boot logs. These logs can also be used for remote

attestation of the platform before boot. TPM is also used to provide cryptographic store for Microsoft's BitLocker drive encryption capabilities. In different distributions of Linux operating systems, the concept of measured boot is still under development [34].

2.5.3 Secure Boot in Embedded Systems

Unlike the PC market, embedded systems have a constrained environment for applications that are allowed to execute on a platform. Embedded systems and IoTs designers are much involved in the software development for the application stack. As such, it allows system designers to be more flexible in their approach. A group of researchers has proposed use of special architecture for implementing on chip security[35]. This work is mainly focused towards real time systems and allows secure loading of tasks, secure communication between processes and the mechanisms for local and remote attestation. TrustLite architecture extends the security to runtime by providing a hardware based exception handling and recovery mechanism [36].

As IoTs and embedded systems are designed to perform a specific set of tasks; this allows designers to perform control flow analysis of the entire system. Control Flow Integrity (CFI) analysis is one such static code analysis technique that records all possible branches a code can take. Using CFI, advanced code-based attacks such as Return Oriented Programming (ROP) can be mitigated. Using hardware based trusted isolated execution environments such as ARM's TrustZone, CFI can be implemented on an embedded system, where jump to each branch can be verified prior execution [37]. Hardware Performance Counters (HPCs) are special registers that count the occurrence of hardware events. These counters are used in performance tuning of applications. ConFirm architecture uses the HPC counters to count triggered events embedded in the control flow graph. CFI is ensured by use of these HPC counters[38].

There are also commercial embedded systems specific solutions for implementing

secure boot. NXP offers secure boot for their architectures. Their secure boot mechanism called High Availability Boot (HAB) is a part of the on-chip ROM. HAB provides a root of trust to the remaining software components running on top of the device. It can provide just signing or both signing and encryption for the software starting from the First Stage Boot Loader (FSBL). Efuses on a device must be configured to use the feature. Private keys are also stored on the efuses. Which means that one the keys have been generated, they cannot be changed. However, the system allows for up to four SRKs to be used. If needed, a SRK's use can be revoked. There is also support for open-source second stage boot loader U-boot for this technology to extend support to the Operating System, such as the Linux Kernel[39].

2.5.4 Secure Boot in Reconfigurable Computing

Commercial FPGA vendors have increasingly incorporated new security features over the years, but that is not consistent across the reconfigurable devices and does not include legacy devices. These security features target various aspects of security, such as secure boot, encryption, and data integrity. Here, we give an overview of those features targeted towards Secure Boot. Zynq 7000 FPGAs, the FPGA SoC provides an AES 256 based encryption engine. Additionally, RSA asymmetric authentication is used to ensure an authenticated source. These functions are implemented as a hardware-based function on the FPGA fabric and are part of EDA tools with no access to the end user [24]. Both the encryption and authentication processes use keys to be defined before deployment. In the Zynq 7000 architecture, there are two ways of key storage, namely Battery-Backed RAM (BBRAM) and one-time programmable fuses. BBRAM is an on-chip volatile memory region, that has to be battery powered. One-time programmable efuses are configurable fuses that are burned into the fabric once they are programmed. There are caveats of using these technologies for holding keys. The BBRAM is dependent on the presence of a dependable power source. In the case there is any fluctuation in the power supply, the keys will be lost. On the

other hand, the efuses are once programmable. In the case the programmed keys leak, there is no way to refresh them. Another serious flaw is that these keys are stored unencrypted in the memory. Access can be gained to them through physical means and through side channel attacks.

These cryptographic processes are invoked as part of the secure boot process implemented in the Xilinx provided zeroth stage boot loader called BootROM[24]. BootROM code exists in non-volatile memory on the FPGA fabric. It is executed by the Processing System on FPGA at the beginning of the boot cycle. Once the execution is completed by the BootROM code, control is passed to user-defined code, which can be an operating system or a user level application. The vendor provided no access to the code implementation of BootROM and neither is there any read or write access to the memory holding the BootROM code.

Much recently, in the newer Zynq UltraScale architecture, Xilinx has built upon the schemes in the Zynq 7000 architecture to add more security solutions[40][41]. The major revision is in the addition of Physical Unclonable Function (PUF) based key support. Whenever keys are generated for encrypting bitstreams, they are given as input to the on-board PUF to generate an encrypted key. This key can be stored onto the efuses, the BBRAM or externally in any unencrypted space. Since an attacker does not have access to the PUF implementation, they will not be able to decrypt the encryption key. Once the host system sends an encrypted bitstream to the FPGA board, the encrypted key is decrypted at runtime to reveal the bitstream decryption key. This key is used by the encryption subsystem to decrypt the bitstream.

Recent work on the secure bitstream configuration at the boot level proposes the use of PUF technology and on-board peripherals for FPGA bitstream secure boot [42]. Internal Configuration Access Port (ICAP), that is a programmable interface[1], is used to retrieve the configuration of the current programmable logic. A Hardware-Embedded Delay Physical Unclonable Function (HELPUF[6]) uses SHA3 digest as

the challenge input. The generated output is used as a key for decrypting the image of the operating system and the application software to realize self- authentication. This enables the second stage boot loader to program the programmable logic PL and processing system PS. Since, vendor-provided secure boot is not used with this system, ICAP also allows readback of on-chip memory elements such as the block RAM and registers. If an adversary can capture the readback process, they will also be able to read the PUF output as well and therefore will have access to decryption keys.

Another recent work implements self-authentication of the logic fabric design and extends the protection to include processor design[43]. As opposed to [42], this work has employed the use of Elliptic Curve Cryptography based asymmetric keys with Diffie Helman key exchange for the key generation for encryption. A fuzzy key extractor is used to use extract hardware-based variations and to generate a key. This extracted key is used in the design by various cryptographic functions such as remote attestation and encryption key generation.

Dynamic Partial Reconfiguration (DPR) allows reconfiguration of pre-defined sections of the FPGA fabric during runtime. Static design reconfiguration requires the FPGA to be shut down before it can be reprogrammed. Following current security standards followed by FPGA manufacturers[44][24], the static encryption key must also be shared with the third party. Thus, adding more actors who would have access to the key. In [45], authors improve the security with the integration of hardware vendors in the distribution process. Each end device has a Unique ID. This ID is exchanged with the IP provider during an IP purchase transaction. The IP provider passes the unencrypted IP bitstream and the obtained ID of the end device to the vendor. The vendor has a database of IDs of end devices and their corresponding encryption keys. It encrypts the bitstream using the end device's stored encryption key before it is transmitted to the device. This solution does limit the issue of dis-

tribution of the encryption key between parties, however, promotes the use of static symmetric encryption keys. Attacks on vendor provided security, such as Xilinx's Secure Boot have also been reported. Outsourced trusted third-party IPs, based on their application also have access to the main memory of the system. Since the main memory is also used by secure boot to store the unencrypted Second Stage Boot Loader (e.g. U-boot) used by the system, a hardware trojan can target the main memory to manipulate the Second Stage Boot Loader to execute a malicious executable. [46] suggests wrapper logic to be added to IPs before the IP is connected to the system bus. Hardcoded configuration of the wrapper defines the scope of memory access by the connecting IP.

2.6 Logic Locking

Logic locking is a technique for combating IP piracy. In the semiconductor industry, the chip designer has outsourced the manfucturing process to offshore untrusted foundaries. Because of the costs involved in manufacturing, there are separate industries that deal with design and manufacturing. Chip designers outsource their fabrication needs to these companies. For the IP design owners, it is difficult to monitor their IP at a remote site. Their IP may become a prey to piracy in the form of overbuilding, reverse-engineering and cloning.

Logic locking is a design for security mechanism that adds a lock into the circuit. Gates with key inputs are placed at various locations in the target circuit. Until the correct input key is not given as input, the circuit will not produce the correct output. Logic locking can be incorporated into the IC manufacturing process to mitigate IP piracy. [47] is one of the earliest solutions proposing use of logic locking via combinational locking. A combinational lock is embedded into the fabrication process. The key is kept hidden from the foundry and is only punched into the design, once the fabricated chip is returned to the vendor. The foundry not having access to the key is not able to unlock the chip. This method was defeated by [48],

as it notes that the method presented in [47] uses testing mechanisms that give out details of the implementation. [48] presents methods that factor the test pattern inputs. Logic locking is seen as a satisfiability problem. An attacker having access to the inputs and the system's expected responses, can use a satisfiability solver to solve for the key input values. This is known as a SAT attack[49]. Researchers were able to find the keys for 95% of the 441 locked circuits considered.

More recently, work has been done to make circuits resistant to SAT attacks. SARLock[50] is a mechanism that adds complexity by adding a masking block. If the key input cannot be asserted against a system input, the output is flipped. Therefore, the circuit cannot be probed without the correct key input. An alternate method to counter SAT attacks is Anti-SAT[51]. This method adds complexity to the output by adding another layer of logic. For incorrect key input, the output regardless of the expected output will either be a 0 or a 1 depending on the inputs. This way, an attacker will not be able to ascertain if the correct key input is given or not. [52] presents an attack mechanism for AntiSAT and SARLock. This work proposes to reduce the complexity of ANTI-Sat or SARLock locked circuit to the level where it can be solved by a SAT solver. It does by trying to find random key inputs that may reduce the output corruptibility for a system input. A recent approach towards implement logic locking is Stripped Functionality Logic Locking (SFLL)[53]. In SFLL, parts of the logic are removed from the logic circuit in a controlled manner. These removed parts are expected to be added to target device in the form key input dependent circuit.

CHAPTER 3: Boot Time Security and Over-the-Air Update Mechanisms For FPGAs

3.1 Introduction

Internet of Things (IoT) are ubiquitous devices that have limited functionality and computational resources with the capabilities of connecting with other electronic devices and the Internet. Additionally, they have long life cycles spanning over years. Changing requirements of a deployed embedded device are generally addressed through software or firmware updates. Software updates can be provided either manually or physically by using cables and programmer interfaces. Physical access to a device for firmware upgrade is always not possible and requires the configuration of the devices in the field through secure channels. The connected nature of IoTs makes them accessible remotely, and firmware updates for devices can be provided using Over the Air (OTA) firmware updates [54][55].

Software firmware updates change the functionality of the software; however, the scope is limited by the hardware capabilities and its architecture. Depending on the application and the expected life of a product, reconfigurable hardware architecture can improve the hardware update needs to evolve an IoT's device architecture. Reconfigurability allows the manufacturer to update the hardware design while the machine is in the field. Using OTA, hardware updates allows for updating the device hardware configuration without the need for physically replacing it. Over the air (OTA) updates are critical in the embedded system consumer domain such as the cellular phone and the automotive industry. The requirements for availability and quality of service is high in the growing and ubiquitous electronics on the move or electronics on the wheel. For the vehicular domain, the requirement for availability is

directly tied to safety. Furthermore, it is not feasible for a car owner to take the car to a service station, whenever there is a software update available. Instead, firmware updates can be provided remotely with OTA updates that can be transferred via the cellular network directly by the manufacturer[56].

Root of Trust is an anchor for implementing trust in a device[57]. Maintaining Root of Trust (RoT) is crucial once a device has been deployed in an untrusted field. Tainted firmware updates can break the trust. The software domain employs various techniques for maintaining Root of Trust. Some popular examples include Universal Extensible Firmware Interface (UEFI) secure boot extensions[58] and Microsoft Windows' Secure Boot[59]. The former provides maintaining Root of Trust at the boot level and the latter extends it to the operating system (OS) level. The root of trust at this level can also be implemented using cryptographic processors, such as Trust Platform Modules (TPM).

With new advancements, reconfigurable hardware has become pervasive in the Internet of Things domain, there is a requirement for extending the root of trust to the hardware. Commercial FPGA vendors provide limited security to the programmable logic fabric and those security mechanisms are limited in application. Additionally, the provided methods have closed access which the users can use in their systems but cannot inspect themselves. There is a need for an open and reliable security structure for the programmable fabric that can be integrated into the device's Root of Trust.

In this work, we present a framework to establish the Root of Trust for secure boot and OTA updates of reconfigurable hardware. Bi-directional or mutual trust is established between an FPGA device and server or the content provider. A scheme for the provision of symmetric encryption keys is also presented. The novelty of this work also extends into the integration of TPM with FPGA boot process to assist in secure boot, key provisioning, and secure communication. Additionally, we present schemes for runtime mitigation of malicious logic insertion.

3.2 Threat Model for Secure Boot of FPGA Bitstreams

The FPGA market is dominated by proprietary tools, Intellectual Properties (IPs) and closed hardware implementation. There are a handful of vendors that provide varying architectures and interfaces to those architectures. It forces reliance on the vendor and the effectiveness of their often, closed source tools. The reconfigurable fabric on FPGAs is programmed using bitstreams. The bitstream configures the Look Up Tables (LUTs) in the logic fabric. These LUTs act as combinatorial logic and sequential data paths for the hardware design. Bitstreams also configure other fabric elements, e.g. on-chip memory, Digital System Processing (DSP), clocking blocks and wire connections. An attack on the bitstream can affect the entire system operation of a device on the field. This work focusses its efforts towards the security of the bitstream on the device and on providing security between a content provider and a device.

3.2.1 Bitstream Spoofing

Bitstream spoofing is a way of updating the victim device with an update that may seem to come from an authorized source. One way for performing bitstream spoofing is by the use of relay and replay attacks[60]. An adversary can act as a man in the middle between a bitstream content provider and a device. Once an authenticated session has been set up between the two nodes, the adversary can replace the original bitstream with a malicious one. Additionally, in case where the victim device is using one single key for bitstream encryption, replay attacks can be used by an adversary. An attacker can forward an older copy of the bitstream which has limited functionality compared to the current version.

3.2.2 Runtime Malicious Modification

Our work focuses on mitigating malicious logic insertion in the bitstream during runtime. Once a bitstream has been programmed onto an FPGA programmable logic

fabric, an FPGA device may provide interfaces to the outside world for readback and modification of the running bitstream[61]. Using these interfaces, faults or trojans can be introduced in the design[62]. Additionally, the same interfaces can be used to make unauthorized modifications to the original design.

3.2.3 Non-secure Communication with Content Provider

For an FPGA device placed in the field, bitstream updates can be provided manually physically by an engineer, through a physical update mechanism or through the use of remote updates over a network. If a content provider over a network is not secure, an adversary may spoof its identity to send malicious updates. On the other hand, an adversary can also impersonate a device on the field to download bitstream updates from a content provider not meant for it.

3.3 Root of Trust Architecture

In the proposed scheme there are two actors, the content provider or the server and client nodes. The content provider is a server that exists over a network for all client nodes. It serves the latest bitstream to all clients. A client node is any authorized FPGA device that is connected to the network of nodes. The authentication and authorization privileges for a client to join the network is open to the implementation and is left at the discretion of the designer. There are two components in this framework, namely the proposed hardware and the Secure First Stage Boot Loader (SFSBL).

3.3.1 Hardware Overview

The reference reconfigurable platform is an SRAM based FPGA SoC that has a hard-core Processing System (PS) and a Programmable Logic (PL) fabric. The PS is a trusted verifier and the PL design is the prover. The verifier is a trusted source, whereas the prover needs to prove its legitimacy to the verifier. We investigate the secure co-processor integration with the SoC and export the security functions and key

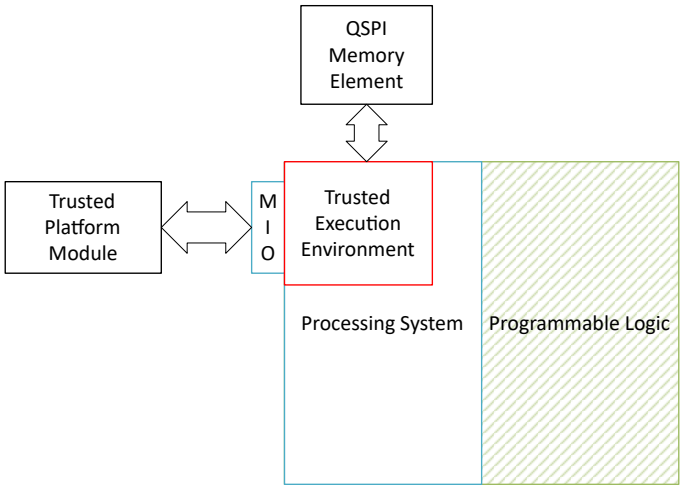

Figure 3.1: Proposed Secure Boot System Architecture.

provisioning onto an external cryptographic processor, such as the Trusted Platform Module (TPM).

To provide isolation for a higher level of security functions that is for Secure OTA update functions, the framework uses a Trusted Execution Environment (TEE), such as ARM's TrustZone. TPM is interfaced using the Multiplexed Input /Output (MIO) port of the FPGA via serial SPI interface on Xilinx Zynq 7000 series FPGA. An MIO port is directly connected to the hard core of the PS and does not require any additional routing from the PL. The MIO based connection to the TPM is essential since in the case PL based configuration is used, the PL will be needed to be programmed first, thus introducing a possible attack vector for the target platform.

To mitigate any damage caused due to privilege escalation attacks, PL programming port and the TPM are configured to be accessible only through TrustZone's secure world. The prototype is tested on the Xilinx platform that incorporates external Quad Serial Peripheral Interface (QSPI) based flash memory to hold the SFSBL and the bitstream. The write access to the flash memory is limited to TrustZone configured secure world. This hardware is described in Figure 3.1.

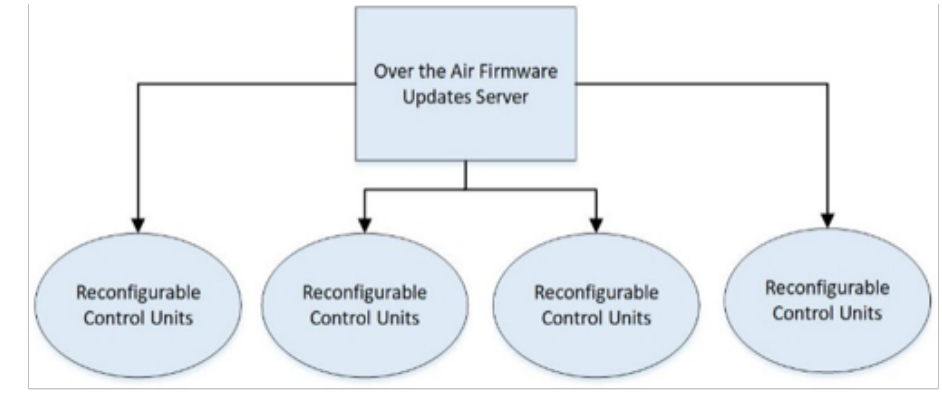

Figure 3.2: Content Provider client FPGA's connection.

3.4 Design Objectives and Operations

The proposed framework is designed to provide three security services for reconfigurable hardware, namely establishing a source of trust, secure over the air updates and secure boot process.

3.4.1 Establishing Source of Trust

A content provider/server may serve multiple client nodes, as illustrated in Figure 3.2. The server is assumed to be secure. To establish the identity of the content provider and a client node, asymmetric digital keys are used. The proposed framework uses Elliptic Curve Digital Signature Algorithm based NISTP256 curve keys [63] for data signing. A pair of asymmetric signing keys are generated on the server as well as each client node. The client nodes use the equipped TPM to generate the key pairs. Additionally, a unique shared symmetric key is generated for each client. This key acts as a base encryption key(K_b) and is updated during the bitstream update process. The update process of the symmetric encryption key is discussed in detail in the following subsections. A client's own private key (K_{cpr}), public key (K_{cpb}), the server's public key (K_{ps}) and the base encryption key are stored on a client's TPM, in its NVM. The exchange of keys occurs in a trusted environment. The server's private

key (K_{rs}) does not leave the system.

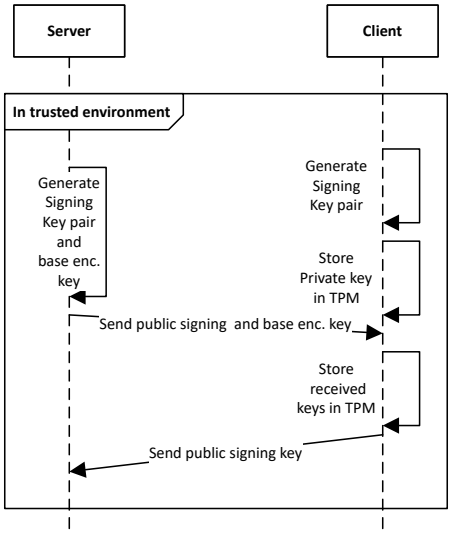

Figure 3.3: Key exchange in a trusted environment.

The server maintains a database of all its client's public keys and the base encryption keys. This system supports key renewal; however, the lifetime of the keys is left at the discretion of the system designer. To mitigate chances for key theft, it is assumed that the renewal process occurs in a trusted environment. Once deployed, the public key of the server and private key of the client cannot be retrieved from the TPM, however, they can be addressed for use within the TPM. The key exchange process is illustrated in Figure 3.3.

Bitstreams are stored in the QSPI Flash on the hardware. The initial bitstream is stored by the vendor in the trusted field. The proposed framework uses Platform Configuration Registers (PCR) to measure the boot process and to maintain prolonged integrity of the bitstream. PCR register, PCR_t is used to store the cumulative SHA256 of all the bitstreams loaded onto the client. Initially, the value of PCR_t is set to the SHA256 of the initial bitstream. The server also calculates the same cumulative hash locally (SHA_t). For use during secure boot at the client, the SHA256 of the bitstream (SHA_c) is stored on the NVM in the TPM. Figure 3.4 shows the keys

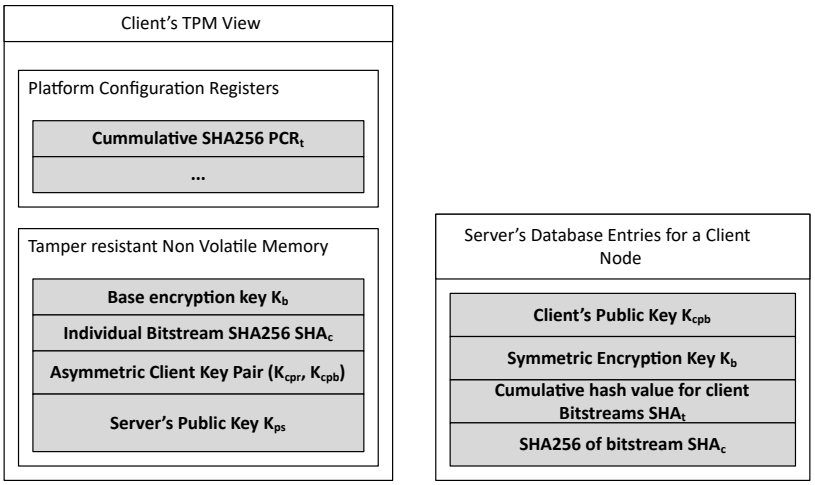

Figure 3.4: Keys shared between a client and server.

on a client node and shared between a server.

3.4.2 Secure Over the Air (OTA) Update Mechanism

When the updated bitstream available on the server, the connected and authorized client nodes receive a notification from the server. This triggers a client to initiate secure communication with the server. At a client, the mode of operation switches from normal mode to the update mode. The update mode is handled by the trusted execution environment at the client. In the case of TrustZone, the processor mode switches from the non-secure world to the secure world. The update process is assigned its own memory area, isolated from the non-secure world. The update procedure consists of the following processes:

3.4.2.1 Secure Communication Handshaking

To enable secure communication between the server and the client, as well to maintain that the integrity of the bitstreams is maintained between two subsequent updates, we propose a handshaking scheme. The scheme is summarized in Figure 3.5. A client generates a True Random Number (TRN_a) using the True Random Number Generator equipped on the TPM. The PCR register value PCR_t is XORed with TRN_a and is then symmetrically encrypted using the base encryption key K_b.

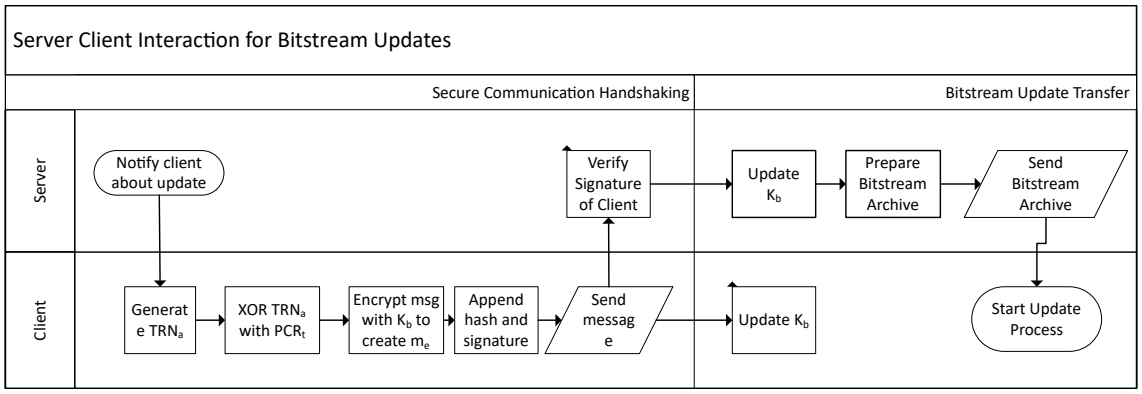

Figure 3.5: Server Client Interaction for Bitstream Updates.

$$m_e = AES128((PCR_t \oplus TRN_a), K_b) \tag{3.1}$$

The construction of the encrypted message (m_e) is shown in Equation 3.1. Hash of m_e is asymmetrically signed using K_{cpr}. Message m_e and its hash is sent to the server. After transmission, the K_b is updated client side by taking a XOR with TRN_a. K_b on the TPM is also replaced with the newly computed value. Equation 3.2 illustrates the update process.

$$K_b = K_b \oplus TRN_a \tag{3.2}$$

The server using its copy of K_{cpb} recalculates the hash of the received message m_e. Once the client has been authenticated, the server decrypts me using the stored key K_b. To retrieve TRN_a, the server computes the XOR of SHA_t with the decrypted message. Like the client node, K_b is updated at the server.

3.4.2.2 Secure Update Packaging and Transfer

The bitstream update is compiled into an archive package before it is sent to a client. Using the bitstream update and SHA_t, the server calculates the new cumulative hash and updates SHA_t. The updated bitstream and SHA_t are combined and encrypted using the updated key K_b. This encrypted block is copied to the update archive

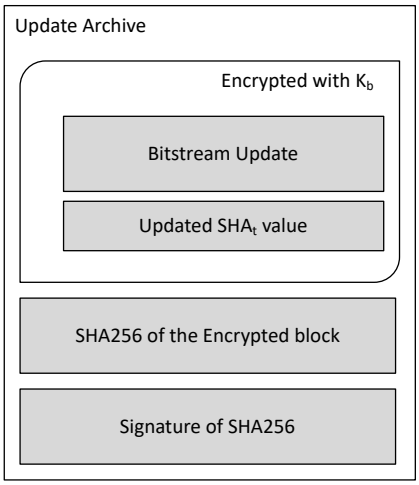

Figure 3.6: Bitstream Update Archive

package along with its SHA256 value. The SHA256 value is signed using its own private key K_{rs}. The signature is appended to the package. This update archive is sent to the client. The client stores the package in its NVM and triggers the update process. Figure 3.6 shows the archive package.

3.4.2.3 Applying the update

The process of applying the update on the target platform is handled by the secure update process running on the client FPGA. Once the bitstream package has been downloaded, SHA 256 of the package is recalculated client side and is compared with that in the download package. The signature of the computed hash is recomputed using the key K_{ps} on the TPM. This signature is compared against the signature packaged in the update archive. Once the identity of the server has been verified, the encrypted archive is decrypted using K_b. Using the TPM PCR functions, PCR_t is updated and verified against the archived SHA_t value. If the updated PCR_t and SHA_t values are equal, the initial bitstream on the NVM is replaced by the update, otherwise if in case any check fails during the update process, the node is assumed to be compromised and the server is notified. On completion of the update process, the individual SHA256 of the bitstream calculated and SHA_c is updated. The process is

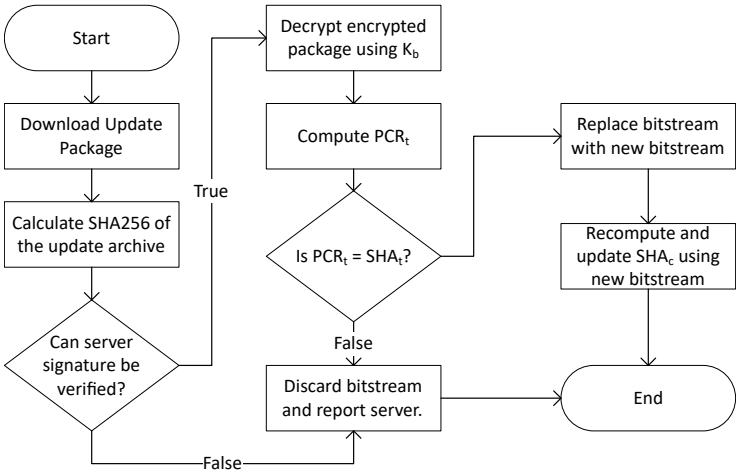

Figure 3.7: Bitstream Update Application Process

described in the flowchart in Figure 3.7.

3.4.2.4 Secure FSBL Boot Process

We have designed a Secure First Stage Boot Loader(SFBSL) that is integrated with the Trusted Platform Module with custom device drivers that enable the device to communicate with the TPM before the device has booted. In an FPGA, the zeroth stage boot loader referred to as BootROM is executed by the Processing System on FPGA at the beginning of the boot cycle. BootROM code exists in a non-volatile memory of the FPGA fabric and is assumed to be secure. Once the BootROM has completed its execution, the control is passed to the proposed SFSBL that exists in the on-board QSPI memory. The SFSBL code extends and integrates the security functions to the FSBL. FSBL configures the device with the hardware bitstream and configures the processing system with a bare-metal application or loads the second stage boot loader to the RAM. This, in turn, loads the operating system. The vendor FSBL design does not verify the integrity of the bitstream or what it is programming the processing system. This lack of security at the boot process can be circumvented with the proposed secure FSBL that integrates key management and security features to validate the configuration files before programming the device.

The proposed SFSBL begins execution with initializing on-board components and memory elements. Once the basic devices have been initialized, the TrustZone is set up. The access to the MIO port connected to the TPM as well as the QSPI ports is both configured to be accessible only by TrustZone's secure world. The secure world code is copied onto the RAM. This segment of memory is only addressable through secure applications. Once the setup is complete, the control is switched to the secure world. The secure world initiates the process of secure FPGA boot. To mitigate exploitation of Time of Check Time of Use (ToCToU)[64] by an attacker, the processing system recomputes the cumulative SHA256 of the bitstream on the TPM. This value is compared with SHA_c stored on the TPM during the bitstream update process. If the values are different, the boot process stalls and the violation is recorded for later reference. Otherwise, the bitstream is configured onto the FPGA programmable logic fabric and the boot process continues.

The secure world sets up service hooks for the non-secure world. These hooks can be used to provide security functions for applications. One such service is the bitstream update service, in which the control is switched to the secure world. The SFSBL process continues with loading the non-secure world application, which can be either a bare-metal application or a second stage boot loader, e.g. U-Boot[65]. The SFSBL process finishes by switching back to the non-secure world and passing the control of execution to the non-secure world.

3.5 Implementation

The proposed framework has been implemented on a Xilinx Zedboard FPGA board equipped with a Zynq-7000 XC7Z020-CLG484[66]. The FPGA has an embedded ARM Cortex A9 hard processor. It is integrated with the Infineon TPM SLB9670, a secure coprocessor for the key management and secure boot processes[67]. The processor is equipped with ARM's TrustZone for Trusted Execution Environment (TEE). Additionally, the on-board QSPI memory is used for holding the SFSBL im-

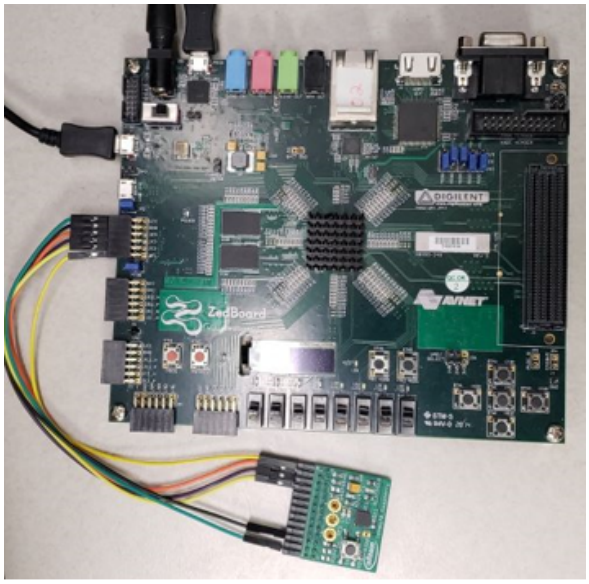

Figure 3.8: Hardware Setup

plementation and the bitstream package extracted from the bitstream update process. Experimental setup of the proposed framework is shown in Figure 3.8.

The FPGA board communicates with the TPM via the Serial Peripheral Interface (SPI) over the dedicated MIO port. The proposed SFSBL implementation is the extension of the Xilinx provided First Stage Boot Loader (FSBL). Secure extensions are added to the existing FSBL that uses the custom device driver library written as part of this research.

The MIO ports are accessible through the secure world configured using ARM processor TrustZone [68]. Total RAM on the Xilinx Zedboard is 512 MB. We have configured upper 64 MB of the RAM space to be used by the secure world. This setting is controlled using the TZ_DDR_RAM register. Additionally, to limit access to the QSPI memory, the QSPI_S_APB bit in the SECURITY6_APB_SLAVES register is set to 0. Typically, for the configuration of peripherals in TrustZone, the value '0' signifies that a peripheral is set to be secure or only accessible from the secure world.

To incorporate the TPM with the FPGA board at the FSBL level, we have imple-

```
int tpm_xfer(uint32_t addr, const uint8_t * out, uint8_t * in, uint16_t len){
    u8 tx_buf[MAX_BUFFER_SIZE];
    u8 rx_buf[MAX_BUFFER_SIZE];
    int transfer_len, ret;

    if (in && out){
        fsbl_printf(DEBUG_GENERAL, "tpm_xfer: in and out cannot both be not NULL.\r\n");
        return XST_FAILURE;
    }

    while(len){
        transfer_len = (len<MAX_BUFFER_SIZE) ? len: MAX_BUFFER_SIZE;
        tx_buf[0] = (in ? 0x80 : 0) | (transfer_len -1);
        tx_buf[1] = 0xD4;
        tx_buf[2] = (addr) >>8;
        tx_buf[3] = addr;

        if (out){
            memcpy(&tx_buf[4], out, transfer_len);
            out+=transfer_len;
        }


        ret = XSpiPs_PolledTransfer(&SpiInstance, tx_buf, rx_buf,4 + transfer_len );
```

Figure 3.9: Secreenshot of tpm_xfer function.

mented a device driver library. This library provides all necessary functions to set up the TPM and implements security functions on the TPM. To the best knowledge of the authors, this is the first library of its kind. The SPI interface accessible through the secure world implements the TPM transfer function driver to communicate with the TPM device. This communication is required at the FSBL level to perform secure boot, which is not supported until the second stage boot loader in the traditional design flow. Figure 3.9 shows the transfer function, that is part of the TPM driver library to establish the SPI interface.

The device driver library is open source and is made available online for public use. The services provided by the library can be divided into two categories, device power-up services, and cryptographic functions. TPM 2.0 architecture has five layers. These layers signify the boot stage for a target platform and are termed as localities. Each locality offers specific functionalities and implements privileges with restrictions of allowed functions, for example, the PCR registers are resources limited to specific localities. Our library implementation provides access to different secure boot specific

```
int tpm_pcr_extend(int loc, u32 index, const uint8_t *digest){
    u8  * pcr_cmd_buf;
    unsigned int offset = 33;
    int ret=0;
    u8 response[500];

    pcr_cmd_buf = malloc(sizeof(tpm2_pcr_extend) + TPM2_DIGEST_LEN);
    memcpy(pcr_cmd_buf, tpm2_pcr_extend, sizeof(tpm2_pcr_extend));

    put_unaligned_be32(33+TPM2_DIGEST_LEN, pcr_cmd_buf +2); //copy digest length
    put_unaligned_be32(index, pcr_cmd_buf +10); //copy index
    memcpy(pcr_cmd_buf + 31, sha256_alg, sizeof(sha256_alg)); //copy sha type
    //copy digest
    if (digest ==NULL)
        return XST_FAILURE;

    memcpy(pcr_cmd_buf + sizeof(tpm2_pcr_extend), digest, 32);
    fsbl_printf(DEBUG_GENERAL, "tpm_pcr_extend: INFO Command Bytes: ");
    for(int i=0; i<sizeof(tpm2_pcr_extend) + TPM2_DIGEST_LEN; i++){
        fsbl_printf(DEBUG_GENERAL, "%02x.", pcr_cmd_buf[i]);
    }
    fsbl_printf(DEBUG_GENERAL, "\r\n");

    ret = tpm_send(loc, pcr_cmd_buf, sizeof(tpm2_pcr_extend) + TPM2_DIGEST_LEN);
    fsbl_printf(DEBUG_GENERAL, "tpm_pcr_extend(): INFO tpm_send() return: %d\r\n", ret);

    memset(response, 0xFF, sizeof(response));
    ret = tpm_recv(loc, response, 500);
    fsbl_printf(DEBUG_GENERAL, "tpm_pcr_extend(): INFO return size: %d\r\n", ret);
    if (ret>0){
        fsbl_printf(DEBUG_GENERAL, "tpm_pcr_extend(): INFO PCR_READ(REG:%d): ", index);
        for (int i=0;i<ret;i++){
            fsbl_printf(DEBUG_GENERAL, "%02x.", response[i]);
        }
        fsbl_printf(DEBUG_GENERAL, "\r\n");
    }
    free(pcr_cmd_buf);

}
```

Figure 3.10: Screenshot of TPM Driver Extend Functions.

functions at all localities. In the proposed framework implementation, the TPM is only accessible from the trusted secure world, the library exists in the scope of the secure world and is not accessible from the non-secure world.

Timing overhead for the proposed solution is dependent on the data rate of the SPI interface and the wait time for each operation. The data rate of the SPI interface is dependent on the host and the TPM device. The TPM2 specifications only specify a maximum timeout for a message transfer and timeout for primitive operations such as requesting a locality and checking the ownership of a locality, etc.

For each message transfer, the TPM can send back a signal for wait state for a reply. The TPM can send a maximum of 100 wait states before a timeout can occur. For the secure boot operation, the TPM structure TPM2_PCR_EXTEND reads data in chunks of 32 bytes to extend a PCR. The bitstream for the Xilinx Zedboard is 3.85 MB in size. Each TPM2_PCR_EXTEND operation takes an input of 32 bytes, it

```
/*****************************************************************************
 *  This function computes the TPM based hash and store it in the TPM on Locality 4.
 *
 *  @param Start Address of bitstream
 *  @param Length of bitstream
 *  @return
 *      - XST_SUCCESS if operation completed successfully.
 *      - XST_FAILURE if operation fails
 *****************************************************************************/
u32 ComputeHashLoc4(u32 StartAddr, u32 Length, u8 * hash){
    //This function assumes that at this point TPM has been set up and initialized.
    int loc = 0; //locality that will compute hash on the device.
    uint8_t digest[TPM2_LOC4_DIGEST_LEN];
    u32 block_start_address = StartAddr;
    u32 image_end_addr = StartAddr + Length;
    memset(digest,0, TPM2_LOC4_DIGEST_LEN);
    memcpy(digest, block_start_address, TPM2_LOC4_DIGEST_LEN);

    fsbl_printf(DEBUG_GENERAL, "Inside ComputeHash hook: \r\n");
    fsbl_printf(DEBUG_GENERAL, "Image Start Addr: %08x\r\n", StartAddr);
    fsbl_printf(DEBUG_GENERAL, "Image End Addr: %08x\r\n", image_end_addr);
    fsbl_printf(DEBUG_GENERAL, "Acquiring Locality 4: %d\r\n",  tpm_request_loc(4));
    fsbl_printf(DEBUG_GENERAL, "Reading Image: \r\n");
    tpm_hash_start();
    for (; block_start_address< image_end_addr; block_start_address += TPM2_DIGEST_LEN){
        memcpy(digest, block_start_address,
            ((block_start_address + TPM2_LOC4_DIGEST_LEN)<image_end_addr)
            ? TPM2_LOC4_DIGEST_LEN : (image_end_addr - block_start_address));

        tpm_hash_data(digest, ((block_start_address + TPM2_LOC4_DIGEST_LEN)<image_end_addr)
                ? TPM2_LOC4_DIGEST_LEN : (image_end_addr - block_start_address));

    }
    tpm_hash_end();
    memset(digest,0, TPM2_DIGEST_LEN);
    tpm_pcr_read(4, 17, digest);
    memcpy(hash, digest, 32);
    return XST_SUCCESS;
}
```

Figure 3.11: ComputeHashLoc4 function computes cummulative hash over the TPM.

takes a total of 126k hashing operations. A snapshot of the TPM extend function from our driver implementation is given in Figure 3.10 for reference.

To reduce the timing overhead for computing cumulative hash in real world applications, TPM 2.0 provides offers a separate locality, locality 4. It allows calling the three structures TPM_HASH_START, TPM_HASH_DATA and TPM_HASH_END. To compute cumulative hash of the bitstream, firstly, the TPM_HASH_START structure is issued. It dictates the TPM to become ready to receive streaming data. Using TPM_HASH_DATA structure, SFSBL streams the bitstream over to the TPM iteratively. Once the transfer is complete, the TPM_HASH_END structure is sent to denote the end of the data input. The computation time was observed to be 40 seconds for the target bitstream file. Fig. 11 shows the implementation of the function. In the SFSBL implementation, this function is made part of the image_mover.c file, since this file is responsible for copying bitstream images between mediums.

3.6 Security Analysis

Reconfigurable computing is becoming common platform for the IoT devices. The primary goal of SFSBL framework is to enable security features to establish secure boot and TPM based over the air secure hardware configuration updates on the field to enable these devices to evolve and update with the change in device requirements, such as security patches. In this section, we discuss the security properties of the proposed framework.

- Digital certificates are used for the identification of the server and participating nodes. The certificates are shared before deployment by the trusted authority and can only be modified by the trusted facility. Thus, impersonation and data spoofing is not possible in the proposed scheme.

- The encrypted configuration bitstream is packaged with the SHA256 of the encrypted bitstream, along with the bitstream package signature signed off with the private key of the server. Any changes in the bitstream will fail the hash comparison and device will discard the updates. This prohibits the man in the middle, and data spoofing attacks and provides an authentication mechanism for securely transfer the configuration files.

- The keys are stored on the tamper-resistant storage inside TPM on server and device nodes, eliminating invasive and probe attacks to break the key. Furthermore, the private keys never leave the premises thus mitigates the man in the middle attack.

- The symmetric encryption key is updated on every bitstream. It ensures freshness of encryption keys as well as mitigates replay attacks. Once the key is used it is never used again and hence any repeated keys can be used to identify attacks.

- During the execution, the device cannot be altered, and the keys can be accessed with the hardware isolation protection of the SPI interface. The TPM interface is only accessed in the secure mode.

3.7 Conclusion

This work investigates the security extensions of reconfigurable logic based embedded devices to enable secure boot processes and the firmware updates that can reconfigure the hardware and software that runs on the device in an untrusted field. We have integrated the TPM capabilities at the boot level with the custom drivers that are integrated at the first stage boot loader to verify the bitstream before the programmable logic is configured or the software is loaded to identify any malicious modifications in the device configuration files. The framework integrates the existing security features such as the trusted execution environment and enables the trusted platform module capabilities with the custom drivers that interface with the FSBL to implement secure boot process. We demonstrate the secure boot process and authentication and process for the over the air updates to the firmware in the field.

CHAPTER 4: Runtime Logic Camouflaging and Obfuscation

4.1 Introduction

Existing work in boot time security for reconfigurable architectures for IoTs [42] [69] [46] present various boot time techniques to secure IPs on the FPGA fabric. However, the proposed techniques all follow similar themes for providing security. Such that the Programmable Logic (PL) space is divided into two partitions, secure and dynamically reconfigurable application partition. The secure blocks consist of cryptographic functions that may provide authentication and encryption functions. Once the device's identity can be verified, the application partition space is reconfigured dynamically.

The application bitstream is either provided at runtime by a backend server, or is stored on the system in encrypted form. This bitstream is decrypted at runtime using response of security functions implemented on the system. To tie the bitstream to the system, existing works have used PUFs to generate device specific responses. The limitation of current work is that they do not provide runtime security for application logic partitions. Once the application logic has been decrypted and programmed onto the fabric, attacker having access to fabric can read off the application design. Thus, resulting in opportunities in bitstream cloning.

This chapter presents techniques for extending security to runtime using logic locking and a novel mechanism of Look-Up-Table (LUT) based bitstream logic obfuscation. It discusses the design and implementation details of how can the PL fabric be interacted, its architecture and scheme for implementing runtime logic obfuscation and camouflaging.

4.2 PCAP Programming

As previously discussed in Section 2.2, Processor Configuration Access Port (PCAP) and Internal Configuration Access Port (ICAP) are used to interface with the PL fabric. There are a number of alternative ways using which the fabric can be configured including SelectMap, Serial, SPI and JTAG [1]. However, PCAP provides an on-chip method. The PCAP hardware becomes available to the on-board processor at boot.

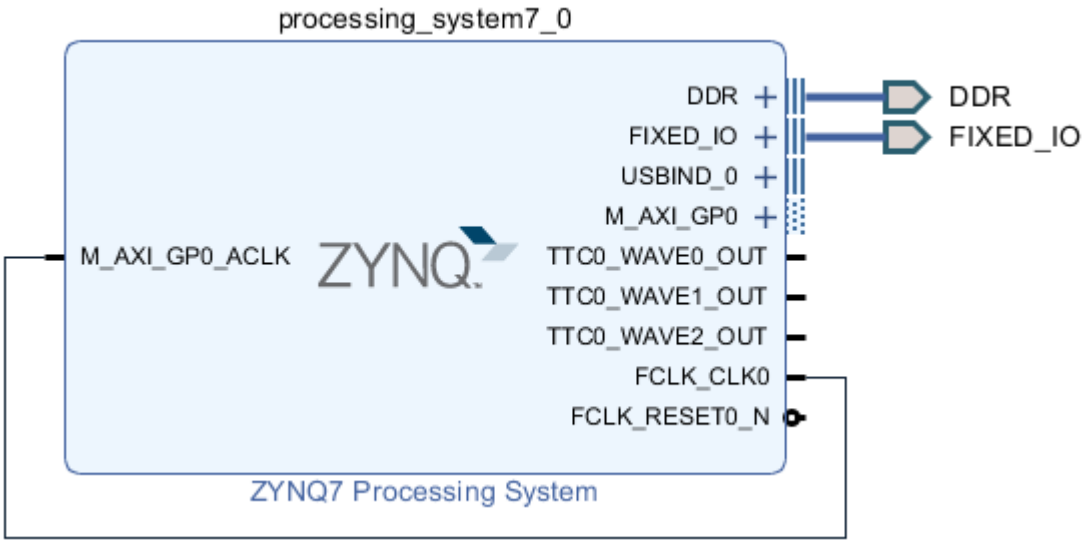

Figure 4.1: Vivado Processing System Block Diagram.

Communicating with PCAP is performed using 32-bit words formed into packets. There are two types of packets, 1 and 2. Packet type 1 has the length of a single 32-bit word.

Table 4.1: Xilinx PCAP Type 1 Packet [1]

Header Type	**Opcode**	**Register Address**	**Word Count**
[31:29]	[28:27]	[17:13]	[10:0]
001	xx 00 - NOP 01 - Read 10 - Write 11 - Reserved	xxxxx	xxxxxxxxxx

Packet 1 is described in Table 4.1. Header type specifies that it is a packet of type

1. There are four kinds of type 1 opcodes, out of which three are usable. In a read and write packet, the number register address specifies which address to read from / write to, and word count specifies the number of expected words. However, a write packet expects a following type 2 packet as input for multi packet input. Type 2 can only be sent after a type 1 packet. Table 4.2 illustrates the packet formation.

Table 4.2: Xilinx PCAP Type 2 Packet [1]

Header Type	**Opcode**	**Word Count**
[31:29]	[28:27]	[26:0]
010	xx	xxxxxxxxxxxxxxxxxxxxxxxxxx

There are several configuration registers that are a part of the ICAP interface. The registers related to this work are discussed here.

- **CRC - Register Address 00000:** The CRC register holds the checksum of the bitstream that exists on the fabric.
- **FAR - Register Address 00001:** Also known as Frame Address Register. FAR is the addresses resources on the PL fabric. Details of FAR are covered in Section 4.3
- **FRDI - Register Address 00010:** Frame Data Register, Input Register is a register which accepts inputs data to the PL fabric. Writing to FDRI is set up using Write Configuration (WCFG) command.
- **FDRO - Register Address 00011:** Frame Data Register, Output Register is a readable register, which is used for reading data from the PL fabric addressed by FAR. Fabric data can be read following sending a Read Configuration (RCFG) command.
- **CMD - Command Register 00100:** Command Register is used to send to perform set actions to the fabric.

4.3 FPGA Bitstream Architecture

4.3.1 Bitstream Contents

FGPA bitstream consists of initialization configuration of logic gates, logic and clock routing, interconnects and hard blocks. Starting from a high level design using hardware descriptive languages such as Verilog and VHDL, the synthesis tool generates an RTL output. This RTL output consists of netlist of the gates used in the design. In the implementation phase, Place and Route (PNR) process is applied. In this process, gates present in the netlist are placed on the PL fabric using resource placement algorithms.

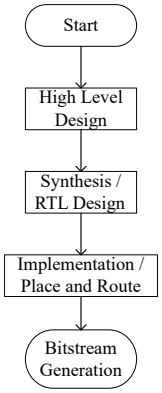

Figure 4.2: Bitstream Generation Flow Diagram

With the resources placed, routing is performed between components. This process tries to generates a placement policy with the constraints defined. Based on the performed implementation, the bitstream is generated. The bitstream output consists of sequence of commands sent to the ICAP. These commands initialize the PL fabric, set up and perform the transfer and start up the fabric to execute the programmed design. A snapshot of the bitstream is given in Figure 4.3. First, the width of the data transmission is configured with the ICAP. The bitstream suggests performing an automatic width detection, as shown from location *0x90* - *0x97*. The connection is synchronized by sending the word *0xAA995566*. This bitstream performs additional initialization actions, including setting up the watchdog timer, clock frequency for

data transfer, checking the model of the FPGA etc., (*0xA4* - *0x13F*. At address *0x140*, the FAR register is set to address *0x0*. The WCFG command is executed to ready the FDRI register to accept configuration data. The configuration length is set at address *0x158*. Figure 4.3 shows bitstream of Zynq 7010 FPGA. The amount *0x5007F0A0* corresponds to 1,342,697,632 words. This is followed by the bitstream sequence and the fabric bring up sequence (address: *0x1FC3DC*).

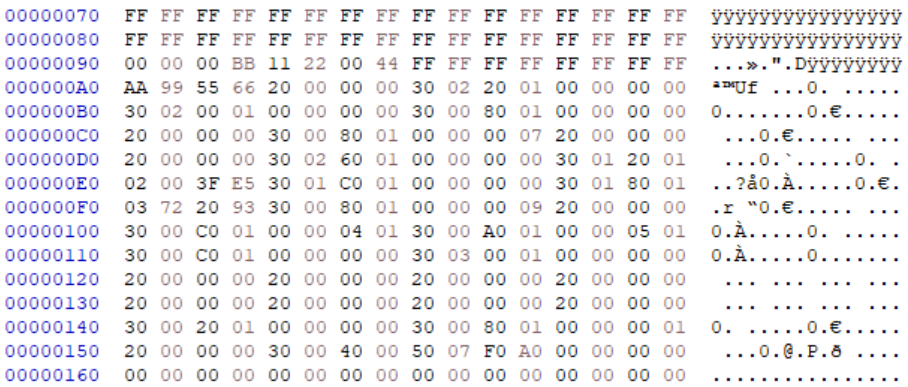

Figure 4.3: Bitstream Header Snapshot

4.3.2 FAR Addressing

Frame Address Register addresses resources on the FPGA fabric. The size of a frame depends on the FPGA model. For Xilinx 7 Series FPGAs, the frame size is fixed to 101 32-bit words. The FAR address itself is divided into segments based on the spatial resource division on the fabric. The division is as follows:

- **Bits [25:23]:** There can be three valid values. *000* for CLB, I/O and clock resources, *001* for block RAM and *010* for CFG_CLB for per CLB configuration.

- **Bit [22]:** The fabric is divided into two regions, top and bottom. This bit selects between the two global clock regions.

- **Bits [21:17]:** These bits are for row selection. The zero point is the center of the fabric, where the two halves meet.

- **Bits [16:7]:** Column selection bits select the column. The addressing starts from 0, from the left of the fabric and increases with ascending columns.
- **Bits[6:0]:** These bits represent the minor addresses within a resource.

4.3.3 Reading and Writing to the PL Fabric

Algorithm 1: Generating FAR list

```
Result: FAR list
frame_count = 0;
frame_array = [];
current_address =0;
last_address=0;
read_word_length = frame length;
Set FAR=0;
while (frame_ count AND 0x38000000) != 0x02000000 do
    Add Shutdown to request;
    Add RCFG (Read Configuration Command) to request;
    Set RCFG request length to read_word_length;
    Execute request;
    current_address = current FAR value;
    if current_ address = last_ address then
        read_word_length = 2 x (frame length);
    else
        read_word_length = frame length;
        frame_array[frame_count++] = current_address;
    end
end
```

Xilinx does not provide a list of addresses to the PL. On examination of the bitstream, as mentioned in Section 4.3.1, the FAR address is initialized to address *0x00000000*. For each continuing word that is written to the fabric, the address increments automatically, once the end of the frame is reached. This behavior presents a challenge for reading and writing to individual frames, due to two reasons. As observed, the FAR address does not include the entire 32 bits range of addresses. It only includes addresses that are valid to address a resource as mentioned in the Section 4.3.2. Additionally, single step incremental addresses stop incrementing once

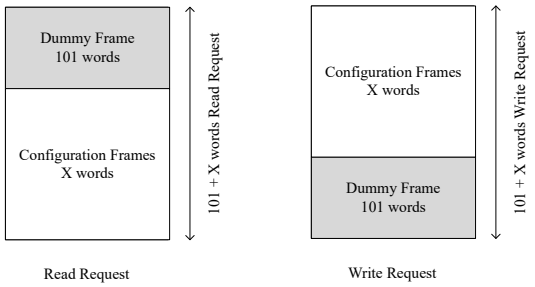

Figure 4.4: FAR Read / Write requests.

the end of row is reached. For example, a valid range of addresses is from address *0x0* to *0x29*. The next valid address is *0x80*. The address range in between the two points is not valid.

Algorithm 2: Performing Readback from the PL fabric

Result: Frame data for FAR address A
frame_data =[];
Initialize request packet;
Set FAR = A;
Add Shutdown request;
Add RCFG to request;
Set read length to two frames;
Execute read request;
Discard first frame;
frame_data = 2nd frame data;

We designed algorithms to capture the entire valid address range. ICAP allows reading configuration data from the fabric, once it has been configured. This process is called *readback*. We used readback to traverse through the address space. During the readback process, once the end of a row in reached, the FAR register does not increment. Once this condition occurs, we can perform a readback of two frames (2 x 101 words) in a single request. This forces FAR to jump to the next valid address. This process is repeated until the address can no longer increment. For Xiling Zynq 7010, this boundary was found out to be address *0x02000000*. The algorithm is illustrated in Algorithm 1.

Once the address list has been generated, direct read and write operations can

```
entity LUT5test is
  Port (
  in0 : in std_logic;
  in1: in std_logic;
  in2 : in std_logic;
  in3 : in std_logic;
  in4 : in std_logic;
  out0 : out std_logic
   );
end LUT5test;

architecture Behavioral of LUT5test is
attribute keep : string;
attribute dont_touch : string;
attribute keep of LUT5_inst: label is "true";
attribute dont_touch of LUT5_inst : label is "true";
begin
LUT5_inst: LUT5 generic map
    (INIT => X"ffffffff")
    port map(
     O => out0,
     I0 => in0,
     I1 => in1,
     I2 => in2,
     I3 => in3,
     I4 => in4
);
end Behavioral;
```

Figure 4.5: Test LUT5 Instantiation

be performed to specific addresses. However, the read and write operations require adding a padding frame. The position of the frame is dependent on the type of the operation. For a successful read operation, the dummy frame is added to the beginning by the PL fabric and is discarded. On the other hand, for a write operation, a dummy frame is added to end of the request when it is sent to ICAP. ICAP will discard this frame when writing onto to fabric. The algorithm for performing the readback on a single frame is given in Algorithm 2.

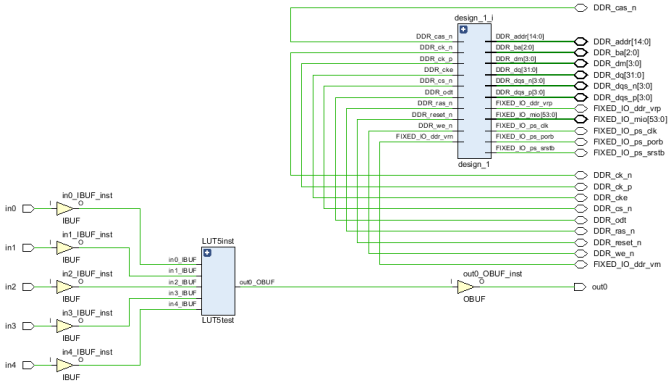

Figure 4.6: Experimental design for evaluating bitstream mapping

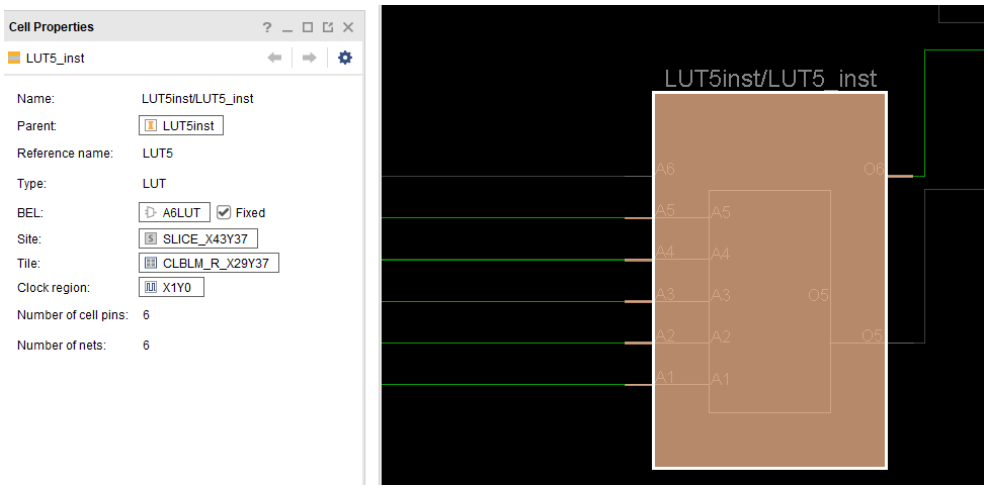

Figure 4.7: Experimental Setup for LUT5 placement.

4.3.4 Mapping FAR to Resource

To study the mapping, a single LUT based design was used. Look-Up-Tables (LUTs) are the building blocks of combinational logic. Xilinx Zynq architecture offer a six-input two output LUT primitive. There are four LUTs per a slice within a CLB. These LUTs can further be programmed to implement one to five input LUTs. In our experiment, we configured a single five input LUT. In order to have access to the PCAP interface the on-board hard processor was instantiated. To create a maximum bit difference on the resultant bitstream, the LUT5 was set to its maximum initialization value, *0xFFFFFFFF*. Additionally, to ensure that the component is not optimized out from the bitstream by the PNR tool, additional attributes were set during its initialization. The definition of the instantiation is given in Figure 4.5 and the resulting system design is given in Figure 4.6. To allow for reliable analysis, the LUT was placed and fixed at the tile location CLBLM_R_X29_Y37, LUTA, as given in Figure 4.7.

Definition for a CLB is divided into multiple frames. Examining reverse engineering mapping, it can be observed that resource information is divided into rectangular blocks, in the form of multiple rows and columns[70] [71] [72] [73]. The size of the

memory region taken by a resource is dependent on the type of resource. For example, a CLBLM_R resource takes 2 words x 36 frames on the bitstream. This block contains information related to LUT configuration, carry blocks, MUXes used to route signals, flip-flops and latches interconnect information. As a part mapping resources to the bitstream, a framework is written for performing readback from the fabric, and querying resource information from the bitstream. Using the framework, a snapshot of the resource CLBLM_R_X29_Y37 of our test bitstream is shown in Figure 4.8.

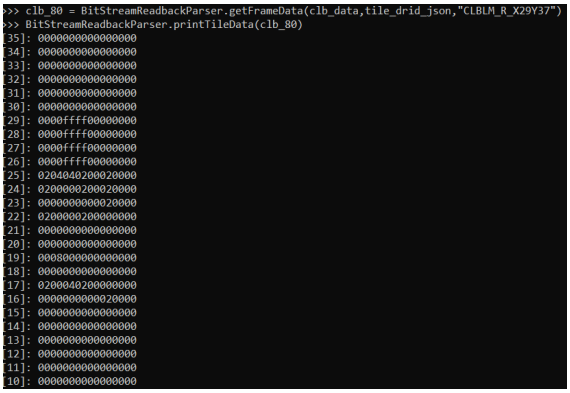

Figure 4.8: Snapshot of the resource CLBLM_R_X29_Y37 from target test circuit.

4.4 Proposed Scheme for Multi-layer Camouflaged Secure Boot

The proposed scheme provides a multi-layer approach for providing secure boot for FPGAs in an un-trusted field [5]. In the field, the device contacts a secure content server placed in a trusted field. The server is assumed to be secure. The application bitstream used by a client is provisioned by the server. Each client device has two memory areas, a Read Only Memory (ROM) and an Application Non-Volatile Memory (ANVM).

As opposed to the existing secure boot solutions, this work proposes use of two bitstreams, namely Attestation Bitstream (ASB) and Application Bitstream (APB). The ASB is composed of a Physical Unclonable Function (PUF) which is used to generate the per-device unique responses, based on the input challenges. APB is the

encrypted logic-locked and LUT camouflaged version of the application bitstream. An overview of the enrollment and in-field operations is given in figures 4.9 and 4.10.

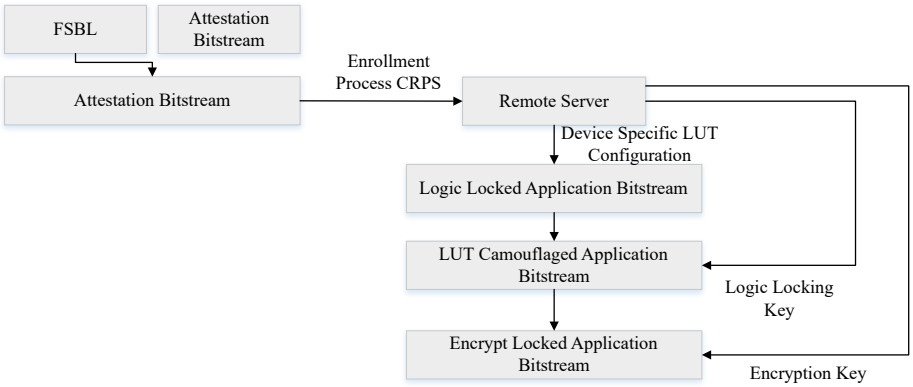

Figure 4.9: Overview of the enrollment process [5].

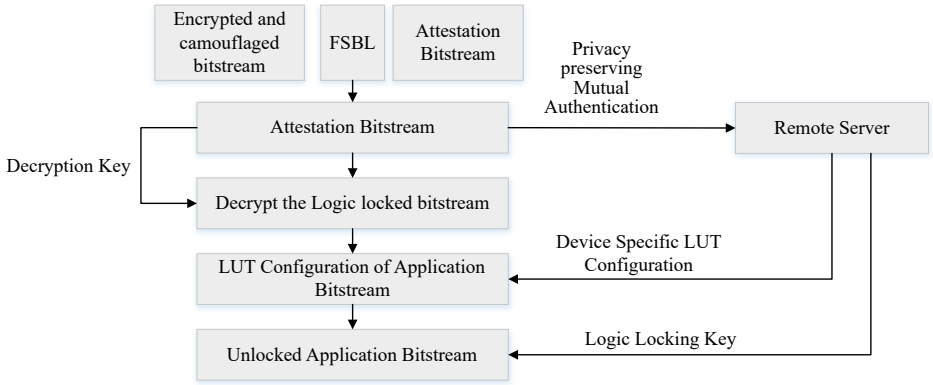

Figure 4.10: Overview of the in-field operation [5].

4.4.1 Device Enrollment

To ensure that only legitimate devices can used on the network, the devices are to be enrolled. Enrollment occurs in a trusted environment. Only authenticated devices can communicate with the server and are eligible to receive updates. ASB consists of the PUF design. Although, the platform is PUF agnostic, for this demonstration, HELPUF is being used. HELPUF implementation generates unique per-device keys [74].

HELPUF is a unique kind of PUF which utilizes functional unit of the design to generate process-variation based delays. This work uses AES encryption unit as the

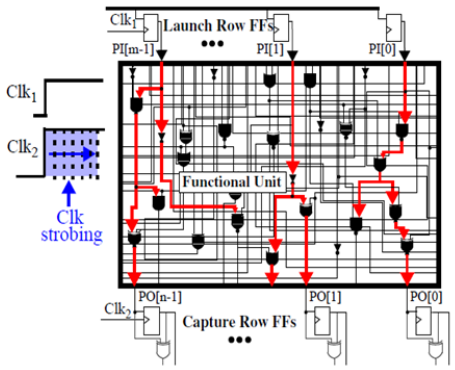

Figure 4.11: Computing Latch-Capture Interval in HELPUF [6].

functional unit. Clock phase differences act as path delays. Input challenges are provided to the PUF controller. The controller translates the inputs to sensitization for the different paths within the AES unit. To capture the delay on each path, a pair of flip-flops, namely launch and capture flip-flops are used. The interval between the triggering and capture is called Launch-Capture Interval (LCI). This interval provides the source of entropy of HELPUF. Additional helper data is required to reliable data. Details of bit generation and mutual authentication using HELPUF is described in [6] and [75].

There are two memory areas on each client device, a Read Only Memory (ROM) and Application NVM. The ROM is once-writable, whereas the ANVM provides a multiple-write persistent storage area. In the trusted field, the ASB is stored on the ROM on the client. The FPGA is booted up and the ASB is configured on the PL fabric. The server has a large set of common challenges stored in a database. It provides the set of challenges serially to the client device and stores it responses against the device ID in database. The server then selects a unique challenge (c), which is stored on the client's ANVM. The response of challenge c, R_s is used as a private key of the key. To maintain on-field anti-tampering measures, the key is not stored on the device itself.

There are two steps required to create the APB. The application design is first

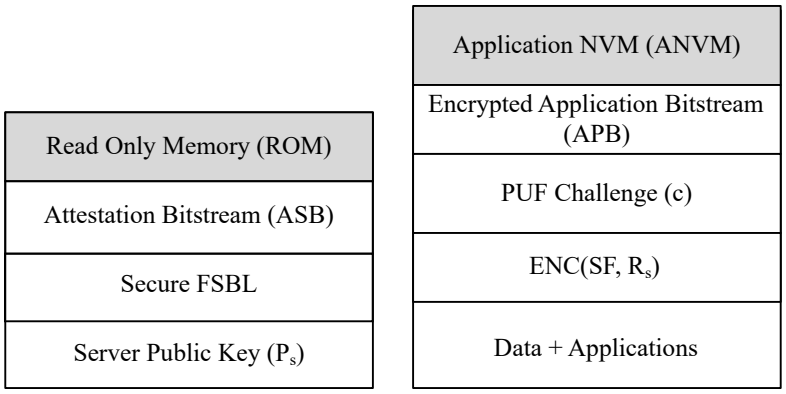

Figure 4.12: Client Device Memory View on enrollment [5].

considered. Using any logic locking technique, a logic locked version of the application design is created. This work proposes a novel logic camouflaging mechanism for bitstream. The server chooses and removes multiple frames from the logic locked bitstream. This list of stripped frames is used to generate an ID (SF) which after being encrypted with R_s is stored on the server against each client device. SF also acts as an ID in the authentication process described in proceeding subsections. This list is also stored on the client FPGA. This generated bitstream then encrypted using R_s, before being placed on the ANVM.

In-field booting is performed using the First Stage Boot Loader (FSBL) software. Since it is the task of the FSBL to load the ASB and APB on the field, it should not be allowed to change over time. Therefore, it is stored on the ROM. To provide digital signing for server-side messages, the server also stores its public key P_s on the client's ROM. For this scheme, RSA is being used, however, any other scheme can also be used [76].

After the enrollment process is complete, the view of the client-side memory is given in Figure4.12 and can now be placed in an un-trusted field.

4.4.2 Device Authentication

The device placed on the field is assumed to have a connection to the back-end server. The connection is required for authentication and APB bring-up. On device

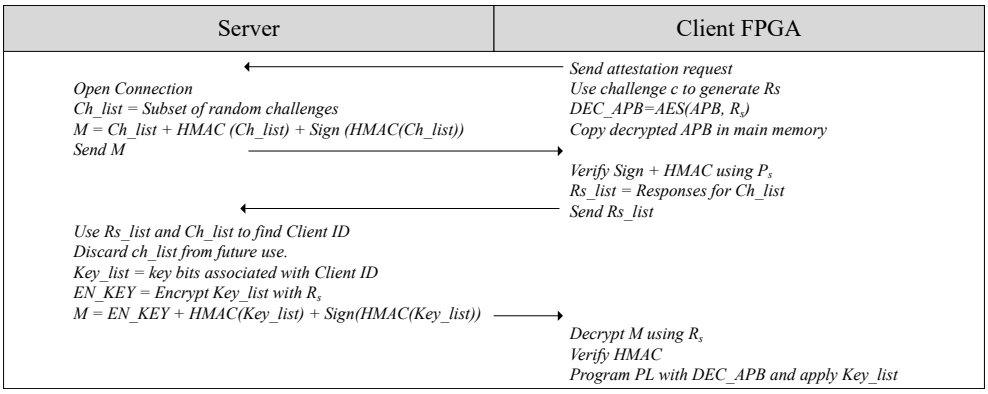

Figure 4.13: Authentication and Application Bitstream Programming [5].

boot-up, the bootROM performs initial device initialization and gives the execution command to the ROM resident FSBL. The FSBL then loads the ASB on the PL fabric. The PUF design encapsulated in the ASB is brought up. Challenge c from the ANVM is provided as an input challenge to PUF. The response recorded R_s is copied onto the main memory for further use. This response is used to decrypt the encrypted APB. The decrypted APB is placed on the main memory. The FSBL shuts down the fabric and configures it with this APB, overwriting the ASB logic present on the fabric.

Authentication of the device with the server is a part of the secure boot process. Once R_s has been recorded, SF is decrypted using R_s and a nonce is XORed to it. The resultant is encrypted again using R_s and sent to the server. For all devices enrolled at the server, the server performs reverse fuzzy matching on the message received to find the correct device. The server then compiles the missing frame information, encrypts it using R_s and sends it to the client device. The server additionally sends key required to unlock the locked design. This key is later used in logic unlocking. The FSBL in the client device decrypts the message and applies the missing frames using the PCAP register. When the configuration is complete, the PL fabric is brought up again.

To unlock the logically locked application, the key received from the server is ap-

```
//Writing frame to PL
CmdIndex = 0;
CmdBuf[CmdIndex++] = 0xFFFFFFFF;    // Dummy Word
CmdBuf[CmdIndex++] = 0xAA995566;    // Sync Word
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x30018001;    //Write IDCODE
CmdBuf[CmdIndex++] = 0x03727093;    //Zedboard ID
CmdBuf[CmdIndex++] = 0x30002001;    // Type 1 Write 1 Word to FAR
CmdBuf[CmdIndex++] = 0x0000239b;    // FAR Address
CmdBuf[CmdIndex++] = 0x30008001;    // Type 1 Write 1 to CMD
CmdBuf[CmdIndex++] = 0x00000001;    // WCFG Command
CmdBuf[CmdIndex++] = 0x30004000;    // Write FDRI
CmdBuf[CmdIndex++] = 0x50000000 | 202; //Write 202 words
for (int count =0;count <202;count++){
     CmdBuf[CmdIndex++] = frame_update[count];
}
CmdBuf[CmdIndex++] = 0x30008001;    //Type 1 Write 1 Word to CMD
CmdBuf[CmdIndex++] = 0x0000000D;    //DESYNC Command
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0
CmdBuf[CmdIndex++] = 0x20000000;    //Type 1 NOOP Word 0

XDcfg_InitiateDma(DcfgInstancePtr, (u32)(&CmdBuf[0]),
          XDCFG_DMA_INVALID_ADDRESS, CmdIndex, 0);

/* Poll IXR_DMA_DONE */
IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
while ((IntrStsReg & XDCFG_IXR_DMA_DONE_MASK) !=
       XDCFG_IXR_DMA_DONE_MASK) {
    IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
}

/* Poll IXR_D_P_DONE */
while ((IntrStsReg & XDCFG_IXR_D_P_DONE_MASK) !=
       XDCFG_IXR_D_P_DONE_MASK) {
    IntrStsReg = XDcfg_IntrGetStatus(DcfgInstancePtr);
```

Figure 4.14: FSBL code excerpt for LUT reconfiguration [5].

plied. If the key is correct, the PL application will start its function correctly, thus denoting a successful boot operation. The authentication scheme is illustrated in Figure 4.13. Figure 4.15 demonstrates modification of LUT data within a frame.

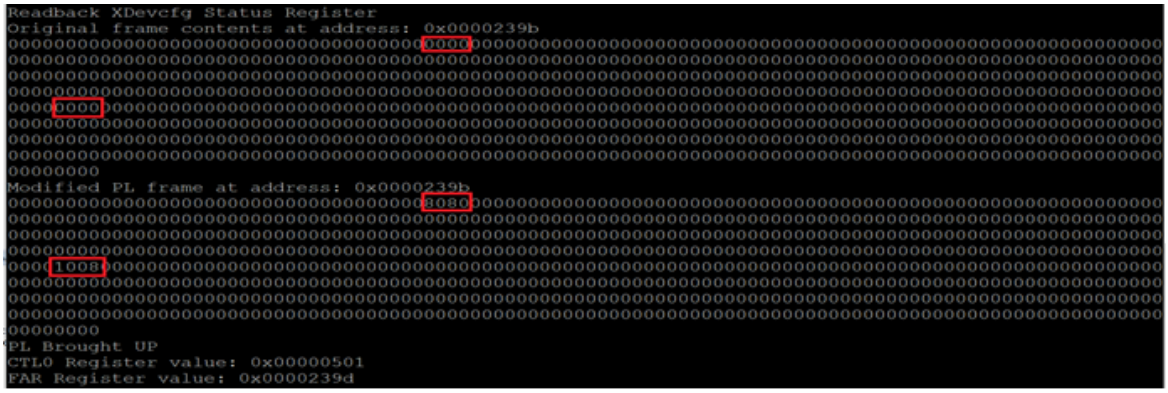

Figure 4.15: On-fabric LUT reconfiguration [5].

4.5 Security Analysis

In this section, the presented framework is evaluated against different attack vectors. Since the presented framework provides a solution for boot time as well as runtime attacks, the security analysis is performed on both avenues:

- Existing works partition the PL fabric space into two. The static partition consists of a cryptographic function. This function can be a Physical Unclon-

able Function (PUF), an encryption unit or a hash function etc., whereas the application is programmed inside a dynamically reconfigurable partition. This introduces secret information leakage issues. If the application PL logic is malicious, or software code running on the system is composed of vulnerable code that can be exploited, an attacker can get access to secrets from the security partition. This presented framework isolates the secrets by dividing the authentication and application bitstreams in two separate full packages. Once the application bitstream is decrypted, it overwrites over the security bitstream. This ensures that they cannot exist at the same time.

- Logic locking and bitstream obfuscation solution guarantees that once the device has been authenticated and has been programmed with the application bitstream, the system can still be made secure. If an attacker gets access to the decrypted application bitstream, they will not have access to the entire bitstream.

- Proceeding de-obfuscation, is application of logic-key for unlocking the logic-locked application design. This extra layer of protection mitigates adversaries which get access to the bitstream once de-obfuscation has been applied. Since the application design requires correct key combination to function, the adversary still will not be able to perform IP cloning attacks efficiently.

4.6 Conclusion

We propose a multilayer secure boot process, that utilizes device level unique physical unclonable function for unlocking the design and updating the LUT frame unique per device to mitigate the security vulnerabilities of maliciously modifying the boot image of bitstream to program the programmable logic. This multilayer secure boot allows the remote attestation server to mutually authenticate, and verify the design running on the fabric with logic locking and LUT frame modification.

CHAPTER 5: Secure Communication Framework for Automotive

We propose a secure automotive framework for Electronic Control Unit (ECU) secure intra communication and code execution

5.1 Secure ECU Communication

The attack demonstrations, enforce the security needs in the safety critical infrastructure[77]. Controller Area Network (CAN) Bus is a multi-master bus where all ECUs broadcast messages over the bus. CAN bus is inherently not secure, all data that is transmitted over the bus is visible to all the nodes connected. The bus requires only a pair of wires namely, CAN Low (CANL) and CAN High (CANH) to form a bus. Devices wanting to connect to the bus only need to connect the CAN transceiver with the bus using these two wires. This bus is linear with terminating resistors at both ends of the bus. A block diagram of the connection is given in Figure 5.1.

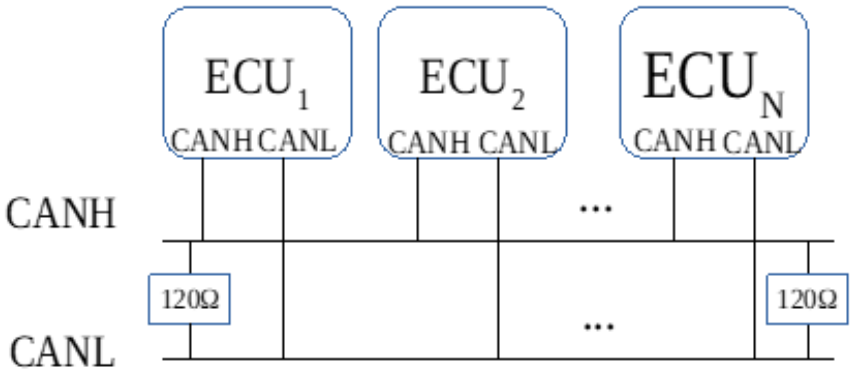

Figure 5.1: Controller Area Network Bus Connection Diagram[2].

[78] and subsequently based on it, its follow-up paper [2], document the weaknesses of the CAN bus protocol stack and propose a hardware based secure communication framework for Electronic Control Units (ECUs) connected to a Controller Area Network. The CAN bus is susceptible to eavesdropping, spoofing and Denial of Service

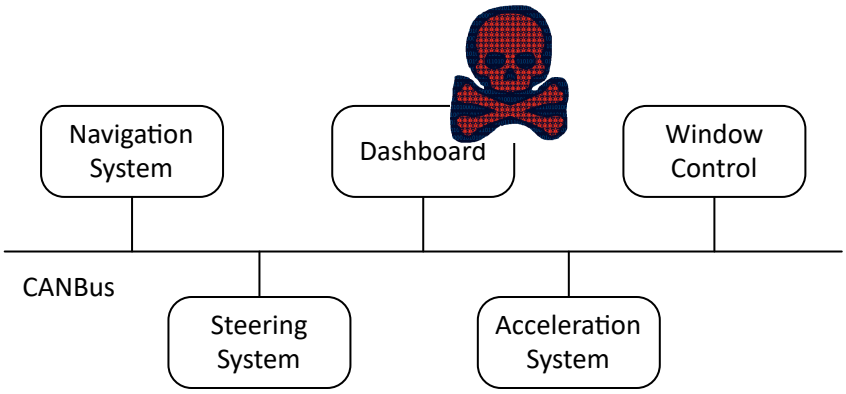

Figure 5.2: CANBus connection in a vehicle.

(Denial of Service).

Table 5.1: Standard Classical CAN Bus Frame

SOF	**Arbitration**	**Control**	**Data**	**CRC Field**	**ACK**	**EOF**
1 bit	12 bits	6 bits	0 - 64 bits	16 bits	2 bits	7 bits

Table 5.1 shows a standard classical CAN bus frame. In a CAN bus frame, arbitration bits are used by an ECU to denote the identity of the sender as well as the priority. All connected nodes listen on the bus. Based on the arbitration ID of each frame received, the nodes may decide to react to a message or ignore it. In the Jeep attack, attackers after getting remote access to the infotainment system spoofed the IDs of the connected systems on the infotainment system. Thereby, the infotainment system appeared as other systems on the bus, e.g. the steering, the brakes, accelerator pedal, etc.

Experimental setup for deriving attack and threat models was set up in lab for this research, as shown in Figure 5.2. It consists of three CAN nodes namely, CAN0, CAN1 and CAN2. CAN0 and CAN2 are considered two legitimate nodes on the network, whereas CAN1 is a rogue node. Figure 5.3 demonstrates CAN1 spoofing ID of another node on the bus and performing a DoS attack. From Figure 5.3, 7DF is the ID of a spoofed node. Any node that recognizes the ID 7DF as a legitimate node on the bus, will accept the message and will react based on the contents of the

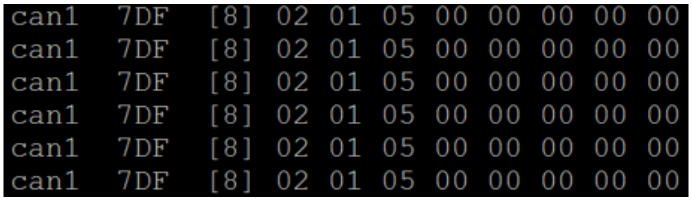

Figure 5.3: Demonstration of spoofing and Denial of Service[2].

message. In the experimental setup, this caused the vendor provided software stack to crash at CAN0, as is shown in Figure 5.4.

From [2], the proposed hardware based security framework is given in Figure 5.5. It uses Elliptic Curve Cryptography (ECC), asymmetric key exchange with Elliptic Curve Diffie Helman (ECDH) and AES symmetric encryption for achieving secure communication between nodes in a vehicle. The framework is equipped on all connecting ECUs and is connected to their processing system in an ECU's System-on-Chip (SoC). The hardware framework consists of five major components, namely, private key generation, key pair generation block, key storage, encryption/ decryption block and shared secret generation block.

There are two phases to the proposed scheme namely, enrollment and deployment. Enrollment is performed in a trusted environment. In case of a vehicle, this can be a factory, a dealer or a trusted service center. There is a resourceful enrollment server that is installed on the vehicle. This server is assumed to be secure and is connected to the same CAN bus as the ECUs. Every component that is installed on a vehicle is first registered with the enrollment server. For the process of enrollment, a challenge input string is first selected for the PUF. This input string is stored in the key storage to be used for authentication. From this input string, the PUF hardware will generate

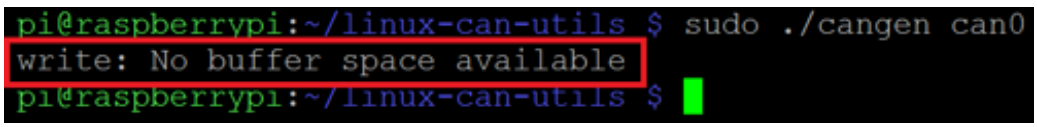

Figure 5.4: RX Buffer at node CAN0 is overflowed[2].

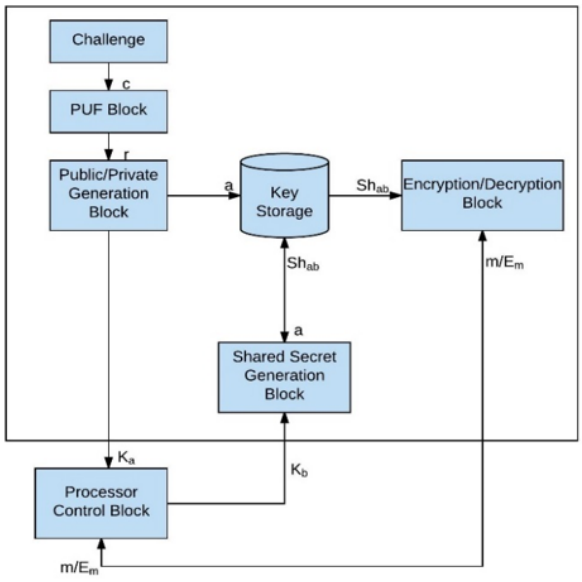

Figure 5.5: Hardware based Secure Communication Framework for ECUs[2].

an output response. This output response of the PUF is used as a private key. The private key is used by the ECC key generation block to generate a public key. The generated public key is stored in the enrollment server. Similarly, the public key of the server is stored in the client's key storage.

On deployment at every boot of the vehicle, the on-board PUF uses the stored challenge input to generate a response. The response is reliable between boots and can be used as a private key. This private key is when used with the ECC based key generation block to generate a public key. Using the enrollment server's stored public key, the ECU's private key and the ECDH module to generate a shared key. This key is used by the encryption / decryption module to encrypt communication with the server. The client constructs a message for the server encrypting its own public key. The server will then wait for all the connected nodes to do the same for a set amount of time. Once the time is over, it will verify the identity of all the connected nodes. This requires the server sending all connected nodes the public keys of the authenticated nodes. If a node cannot be verified, it is blacklisted, and its public key is not forwarded to the other nodes.

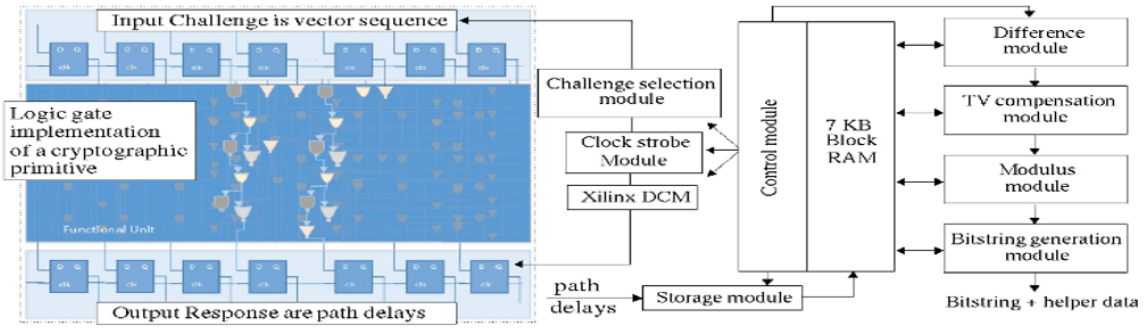

Figure 5.6: HELP PUF Construction[2].

The public keys of communicating nodes are stored in the on-board key storage. The communicating node's public key and a node's private key is used to generate a shared secret key using the ECDH module. This generated key is used in the symmetric encryption / decryption block. Whenever, a message is generated by the processing system, based on the destination, the proposed framework used the generated symmetric key for the communicating node to encrypt the directed traffic. Similarly, the receiving node, based on the sender in the arbitration field in a CAN message, loads the associated symmetric key in the encryption block. The message is decrypted and passed to the processing system block.

Table 5.2: Comparative Analysis of block comparatives speeds at different system clocks rates[2].

	AES-128	**ECC Operation**
At 60 MHz	3666.66 ns	23.8µs
At 100 MHz	220 ns	14.28µs
At 200 MHz	110 ns	7.14µs

This framework enables system wide encryption of the traffic. The public keys of the all the nodes are exchanged in a trusted environment and are stored in the key storage. This mitigates addition of rogue nodes into the system. Additionally, since the system is using PUF for private key generation, it is not stored in any persistent medium. Hardware Embedded Delay PUF (HELP)[79] Block diagram of HELP PUF is given in Figure 5.6. The addition of PUF mitigates the possibility for an attacker for

Table 5.3: Overhead overview at standard CAN connection speeds[2].

	125 kbps	**500 kbps**	**1 Mbps**
Time for sending one frame	864µs	216µs	108µs
During Authentication			
Node sending encrypted message to server (2 frames)	1.728ms	432µs	216 µs
Server's reply (3 frames)	2.592ms	648µs	324 µs

extracting private keys that may be used by an attacker to spoof identities. However, since each standard CAN packet can hold 64 bits, and an encrypted packet is of 128 bits, each message requires two transmissions.

For verifying the design of the framework design, the design was implemented on a Xilinx Kintex KC705 FPGA. The ECC core requires a total of 1428 cycles. With the clock running at 200MHz, the total time to compute a single multiplication point is 7.14 microseconds. In the process for shared key generation, two single point multiplication operations at each node are performed. The ECC core uses a total of 25,454 LUT slices. The AES 128 computation takes 22 cycles to complete, therefore it takes 110 nanoseconds for a complete encryption or decryption operation. It has a footprint of 2560 LUT slices. The generated keys are stored in an on-chip volatile memory, block RAM resources. A block RAM of 128-bit width and 512 elements is instantiated. Key storage area overhead is 8 KB BRAM. This memory element has a read and write speed of one cycle. Table 5.2 shoes operation speeds for ECC and AES operations at different system clock speeds, whereas Table 3 shows the transmission times for sending and receiving messages between nodes and the between a node and the server. This work was awarded Best Poster Award at FICS Conference 2017 in the category of "Hardware Countermeasures"[80]. An extension of this work uses FPGA based Arbiter PUF design for private key generation[81].

Table 5.4: Secure Zone API[3]

Function Name	**Description**
generatePubKeyAuth()	This function is used to generate the authentication message for the server. It contains the public key of the node encrypted using the shared encryption key computed to be used with the server.
addNode()	Once the server responds with an authenticated node, this function is used to pass the encrypted public key of the authenticated node to the secure framework, where it can be decrypted and added to the Key Storage. It also stores the node ID in the normal world for reference.
generateMessage()	This function is passed a node ID of the receiver node and the message to be sent. This function will instruct the framework to encrypt the message using the shared encryption key generated for that node.
decryptMessage()	Once a node sends an encrypted message over the network. This function is called, the sender node ID along with the encrypted message is passed as arguments.

5.2 Hardware based Resource Isolation

The work in [2] and [81] protects the private keys and the entire secure communication process isolated from the rest of ECU on classical CAN. One limitation of this work is that since it uses CAN bus based messages, it is limited by space and date rate of the bus. Each message requires two CAN frames to be transmitted. Additionally, the processing system has direct access to the hardware interface. In case an attacker is able to compromise the processing system, they will have access to the interface of the secure framework. Furthermore, in the previous work, an enrollment sends the ids of all verified nodes to all the nodes across the network. This process not only has a space and network overhead but is also a potential threat. A compromised node having access to all the public keys in its key storage will be able to encrypt data and communicate with all of them.

[3] improves over the secure communication framework to include three major advancements. It uses CANFD, instead on classical CAN. CANFD allows high speed

Table 5.5: Overhead overview at Standard CAN connection speeds with CANFD speed of 8mbps in normal operation

	125kbps	**500kbps**	**1 Mbps**
Node sending encrypted message to server	256µs	76µs	24.42 µs
Server's reply	257.37 µs	77.375 µs	47.375 µs

transfer as well as data payloads to up to 64 bytes. So, only one CANFD frame is sufficient for a message transfer. To reduce the message transfer between each client and the server, on boot the client only receives the public keys of the client it is expected to communicate with, from the server. Additionally, to maintain isolation between the ECU processes and the secure cryptographic IPs and key exchange, this work employs Trusted Execution Environment (TEE). The secure framework can only be accessed through the secure world. To provide an interface to the ECU code, this work presents a set of Application Programming Interface (API) visible to the normal world to handle its communication with the secure framework. A list of functions available are described in Table 5.4. Table 5.5 shows the overhead time incurred for authentication and typical communication operation.

5.3 Secure Code Execution

The proposed scheme detect the malicious intrusion at the time of boot, [7] and during runtime using TPM.

5.3.1 Scenario 1: Code Execution from Read Only Memory

Traditionally, the boot code is placed in the Read Only Memory (ROM) (e.g. BIOS). This code is copied into the RAM at the time of execution. This technique is known as "shadowing". However, with the advancement in high-speed memory technologies and interfaces, it is now possible to execute code from non-volatile memories without the need to copy the code onto to RAM. It is referred to as "eXecute In Place" (XIP) [64]. It is achievable using newer interfaces such as Quad Serial Pe-

ripheral Interface (SPI). Quad SPI allows a higher throughput than the classical SPI. MCUs and the memory can be configured to use XIP mode. Depending on the type of memories and the frequencies of memory accesses, the MCU and the ROM can both be configured to a Random Access or Sequential (Send Instruction Only Once) modes.

In XIP all executable code will exist in the ROM, where the ROM is mapped to the first executable address of the attached MCU. For example, at address 0x0 for a CPU. Depending on the size of the memory, higher address spaces can be mapped to a RAM. The RAM can be used for storing dynamic structures. With this scheme, it can be assumed that the Program Counter (PC) of the CPU can only remain within the ROM address space. With this execution model, any memory access for code outside this region can be considered as illegal access. A hardware monitor is used to check the current value of the PC register. This monitor reads the value of PC at each cycle and it screens each packet and confirms that it is within the ROM address space. With any address access outside this region, the MCU is considered compromised and is logged into the system.

5.3.2 Scenario 2: Hardware-Based/Assisted Core Root of Trust Measurement

In case the executable code is on RAM, it can be modified at the boot time. To avoid the attack during the boot process, the TPM maintains the golden measurements of the trusted executable code. This work proposes a TPM based application code sealing mechanism to avoid firmware update/modify attacks on automotive. In this approach, code sealing is employed before the application code can be executed on the processor.

A TPM provides a set of registers known as Platform Configuration Registers (PCRs). PCRs hold a Hash-based Message Authentication Code (HMAC) digest of the operations performed on a TPM. These register values are changed only using extend operations. Extend uses an old result, for example, HMAC from a previous

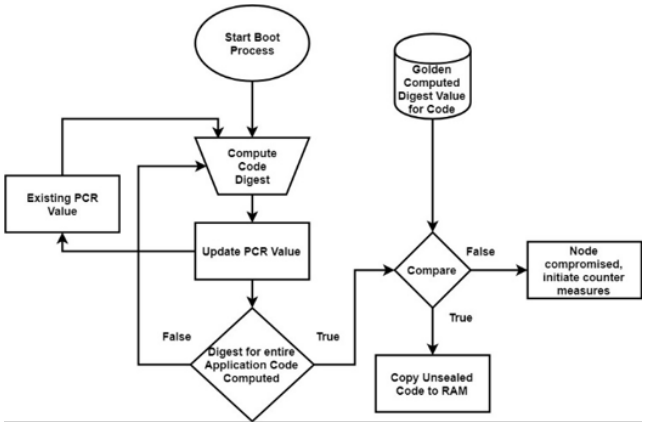

Figure 5.7: On the field code unsealing [7].

operation, concatenates the new result to it and then computes the hash value again. This value is saved to the assigned PCR. By use of this chaining process, the resultant is used as a verification checksum.

The proposed scheme has two phases towards providing CRTM. The first phase is sealing the application code. This is performed in a trusted execution environment. A pair of public and private keys are generated on the TPM. The public key is exported whereas the private key is stored inside the TPM. Using the pair of keys, the application code is sealed. This process requires passing the TPM sequential chunks of the application code, the keys generated and the policy for the PCRs chosen for the sealing process. This process returns the encrypted sealed application code and the computed digest values. All the keys and the digest values are stored in the persistent area of the TPM. These areas are then made immutable with TPM authorization policies. Figure 5.7 describes the measurable secure process in the proposed framework.

5.4 Security Analysis

The security enhanced features of the proposed framework improves the security at the device level and communication over the CAN bus. The framework assumes a

secure server for the enrollment of legitimate ECUs, and the hardware feature HELP PUF allows the detection of any modifications or invasive attacks to the legitimate ECUs, that will corrupt the key generation process and will fail the authentication process at boot-up. Isolation mechanism allows connected ECUs to protect the keys from illegitimate accesses via CAN and does not allow unauthorized code execution.

In the framework proposed, there is no unencrypted exchange of information. An ECU node stores only the public key of the server. No secret or shared key is stored on non-volatile memory the connecting nodes. This makes it resilient against probing attacks and evades the attack where an attacker can retrieve any shared key with physical access to a device and to the NVM.

In the event a node is compromised, the attacker will not be able to retrieve the secret keys to any node. Furthermore, since PUFs are being used to generate keys on each node, all the nodes have different keys. As such, on compromise of a single node, even if an attacker can retrieve a private key from one node, the private key to the other node will still be unknown to the attacker.

In case a malicious node wants to join a secured ECU network; it will fail the authentication process. Before the server can send the public key of this node to the other connected ECUs, it must be authenticated. Since the malicious node's public key is not stored in the server, the server will not authenticate it and therefore will not communicate its ID with the other nodes. In the scenario where an attacker has acquired the public key of a legitimate node on the network, or in case of a server database compromise, adversary can only retrieve the public keys of participating ECUs. Once a communication from the adversary is initiated, the legitimate node will reject it since public key was not initially sent by the server.

5.5 Conclusion

This research investigates the threat models associated with internal vehicular network communication using CAN Bus and propose a secure framework incorporating

security primitives, including PUF, hardware isolation mechanism to protect devices from illegitimate access and hardware based authentication protocols to have trusted and secure communication. The framework is implemented with hardware based authentication and point-to-point encrypted communication for ECUs. The research provides an in-depth description, implementation and performance analysis in terms of timing analysis and area overhead.

CHAPTER 6: Smart Grid Security

Recently, power system's security has gotten much attention due to the various attacks that have surfaced[82][83] [84]. There have been instances of attackers gaining remote access to the power grids causing data/power theft[85] and extreme cases cause power outages, blackouts and physical harm.

6.1 Secure Key Provisioning

[4] proposes a secure framework for key management and secure communication over smart grids using a hardware-based cryptographic processor. The framework protects the confidentiality and integrity of data between each node and additionally provides authentication for the communicating parties.

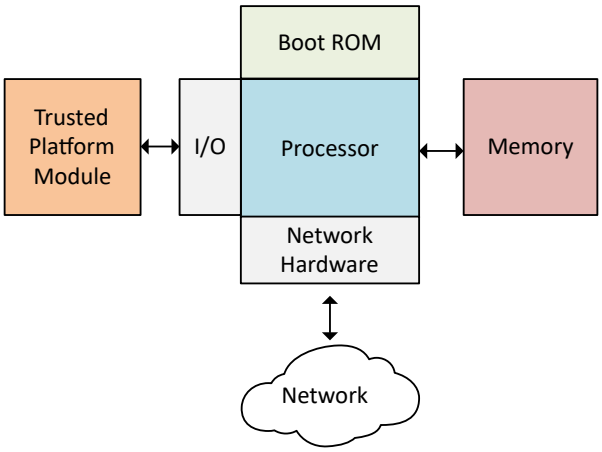

Figure 6.1: Secure architecture for Smart Grid[4].

In the presented architecture, combinations of cryptographic functions are used to ensure not only end to end communication security, but also platform security. The proposed framework assumes multiple nodes to be servicing a target area. Nodes are connected through a network that can be wired or wireless and is left at the discretion

of the designer. For each area which is serviced by a group of nodes, there exists a server that acts as an area Certification Authority(CA) and a master node. The server node is assumed to be secure. Any number of data acquisition or actuation services can be running on the nodes and is dependent on the application and the roles. Each node on the network is equipped with a TPM 2.0 which will act as a root of trust for the node it is attached with. All the nodes hold the public key of the certificate authority in the TPM. The basic architecture at each node is described in Figure 6.1.

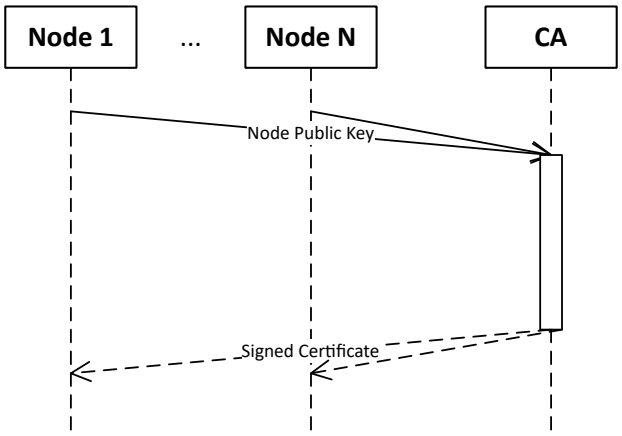

Figure 6.2: Certificate generation[4].

Before a node can be deployed, its private and public key pair is generated in a trusted environment. Simultaneously, the private and public key pair of the area is generated at the CA. The private keys of the nodes and the public key of the area CA are stored on the TPM. To ensure the security of these keys, TPM provides authenticated access to override or to delete existing keys using ownership. Therefore, on deployment, the security of the keys can be guaranteed. Additionally, the public keys of all the nodes are signed by the CA using its private key. The generated certificate is stored at each respective node. Figure 6.2 shows the interaction. Unlike the private keys, there is no requirement to store the certificates in any secure storage, and thus the certificates can be stored in any local storage.

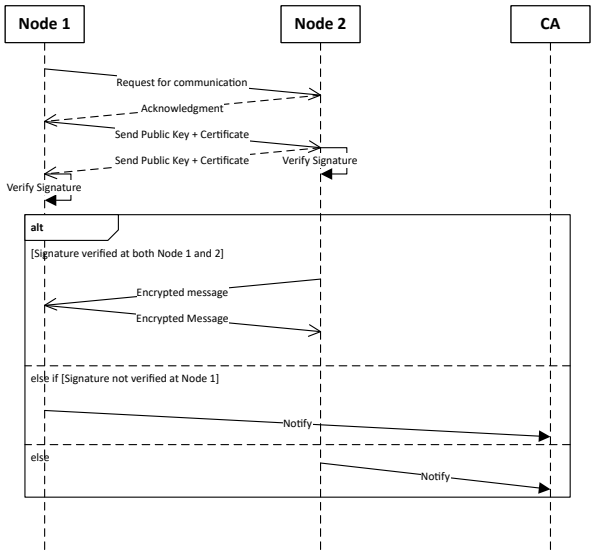

Figure 6.3: Secure communication channel establishment[4].

Once a node has been deployed in the field, to initiate communication between nodes, they both exchange their stored CA-signed certificates. The CA generated signature of the public key of the communicating node is verified against public key of the CA stored in the TPM. The process is illustrated in Figure 6.3.

Once the identity has been verified, the two nodes communicate by encrypting the traffic. For implementing forward secrecy, once the session has been established, both the nodes generate and exchange a pair of ephemeral session keys used for encryption. The public keys are exchanged unencrypted along with the certificates. The ephemeral keys are discarded once the session ends. To deter from replay attacks, each encrypted packet contains a unique nonce, which is also verified by the receiving party before accepting a message. For maintaining the integrity of messages communicated between nodes, SHA256 of the plaintext message is computed on the TPM. This hash is appended to the encrypted message before transmission.

Once the message packet is received on the other end, the message is decrypted, and the integrity is verified. During digital certificate and message integrity verification, if the signature or the hash does not match respectively, the CA is notified. The CA

may notify the backend and blacklist the reported node as compromised, barring it from further communication.

6.1.1 Experimental Setup

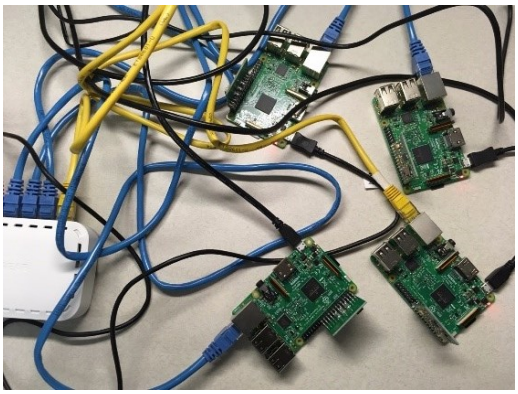

Figure 6.4: Smart Grid test bed[4].

A testbed of Hardware-in-the-Loop (HIL) simulation is emulated with ARM-based microprocessors using Raspberry Pi 3 (RPi3) platform. The nodes are integrated to Infineon TPM 2.0 SLB9670 using SPI interface to add hardware-assisted security to the platform. The configuration of multiple nodes over a network of four nodes was set up, where one of the nodes acts as a master and the rest of the three as clients. A snapshot of the configuration is given in Figure 6.4. Linux kernel versions 4.4 and upwards provide the driver stack for TPM 2.0. Later versions of the kernel have included a kernel-based resource manager, whose task is to provide multiple user access and access context tracking etc.

To implement forward secrecy and to demonstrate TPM's RSA capabilities, ephemeral RSA 2048 asymmetric encryption is used to encrypt the traffic. To generate the primary key of a hierarchy, the TPM2_CREATEPRIMARY structure is used. Based on the primary key created, the TPM2_CREATE structure is used to generate key pairs. Once the key pairs are generated, they are returned to the host machine. This pair is loaded onto the TPM's volatile memory using the TPM2_LOAD structure. To store the pair of keys on the tamper-resistant storage, the TPM2_EVICTCONTROL

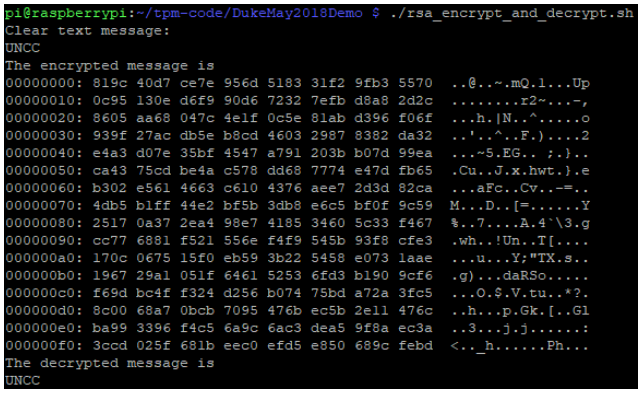

Figure 6.5: TPM based RSA encryption[4].

structure is used. Figure 6.5 and Figure fig:smartgrid-tpm-rsa-enc-dec shows the TPM based RSA key generation and encryption and decryption.

```
pi@raspberrypi:~/tpm-code/DukeMay2018Demo $ cat rsa_encrypt_and_decrypt.sh
#!/bin/sh
#Load rsa keys before running this script.
rm -rf dec_message encrypted_message
echo UNCC > message
echo "Clear text message:"
cat message
tpm2_rsaencrypt -c key_pair.ctx -o encrypted_message message
echo "The encrypted message is"
xxd encrypted_message
tpm2_rsadecrypt -c key_pair.ctx -I encrypted_message -o dec_message
echo "The decrypted message is"
cat dec_message
```

Figure 6.6: TPM based RSA encryption and decryption process[4].

For certificates signing, ECC NISTP256 curve is used for demonstration of its high speed and low overhead (256 bits). The industry adopted transport layer protocol is OpenSSL for certificate generation within the network or by a third-party certification authority. A challenge of using TPM based keys is to be able to use them with OpenSSL. Figure 8 shows the engine integration on the OpenSSL library to be able to communicate the keys generated by the TPM. The engine converts TPM based structures to OpenSSL accessible Distinguished Encoding Rules (DER) and Privacy Enhanced Mail (PEM) formats. Additional tools were created for self-signing certificates on an area CA. The block diagram of this tool is given in Figure 6.7.

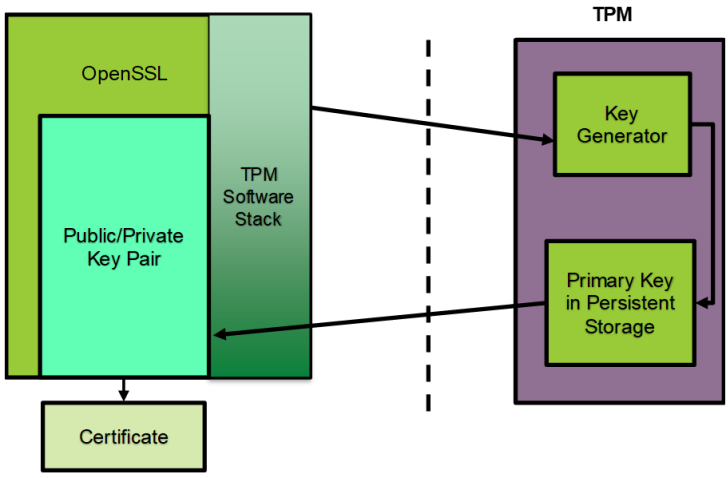

Figure 6.7: TPM based certificate generation[4].

The key ECC parameters are incorporated into the certificate, an example is shown in Figure 6.8. An example of the blob that is communicated to the OpenSSL is shown in Figure 6.9 and Figure 11 shows the certificate generation process at the certification authority using OpenSSL.

```
Prime:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff
A:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:fc
B:
    5a:c6:35:d8:aa:3a:93:e7:b3:eb:bd:55:76:98:86:
    bc:65:1d:06:b0:cc:53:b0:f6:3b:ce:3c:3e:27:d2:
    60:4b
Generator (uncompressed):
    04:6b:17:d1:f2:e1:2c:42:47:f8:bc:e6:e5:63:a4:
    40:f2:77:03:7d:81:2d:eb:33:a0:f4:a1:39:45:d8:
    98:c2:96:4f:e3:42:e2:fe:1a:7f:9b:8e:e7:eb:4a:
    7c:0f:9e:16:2b:ce:33:57:6b:31:5e:ce:cb:b6:40:
    68:37:bf:51:f5
Order:
    00:ff:ff:ff:ff:00:00:00:00:ff:ff:ff:ff:ff:ff:
    ff:ff:bc:e6:fa:ad:a7:17:9e:84:f3:b9:ca:c2:fc:
    63:25:51
```

Figure 6.8: ECC Curve Parameters[4].

6.1.2 Performance Analysis

For this work, a performance analysis was performed, and the results gathered are assembled in Table 6.1. The TPM hardware provides a reliable trust anchor for the host machine. The authors in [86], proposed an RSA implementation which takes 420

```
pi@raspberrypi:~/tpm-code/openssl_engine_example $ cat key.tpm
-----BEGIN TSS2 KEY BLOB-----
MIHSBgVngQUKAqADAQEBoQYCBIEAAAGiWgRYAFYAIwALAAQEYAAAABAAEAADABAA
ILvxy2IlZ7OwOfnlvoaOa04+ySrtyGsIzpsSMFYlTOOEACDDseBF3fghqUH9Ssdq
o6Hs4Pq5Q4bfGGpilFdfJindwQRgAF4AIOVj2KAfkxTg3sWlAc4FxnzHlD5ZfNfb
AHTp8c/bhk2fABC7tdxFdt+mlX4XQhSrJRAfPA/aTE/mnkOEh2bI8k0U3UypElGZ
3sDPLIlkqs8TCABKijwA3tglgDOT
-----END TSS2 KEY BLOB-----
```

Figure 6.9: ECC Key Blob[4].

```
pi@raspberrypi:~/tpm-code/openssl_engine_example $ ./create_openssl_cert.sh
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:North Carolina
Locality Name (eg, city) []:Charlotte
```

Figure 6.10: Certificate generation[4].

milliseconds to compute RSA encryption for 10Kilobytes of data on 32 bit Arduino Mega 2560R3 microcontroller. TPM takes 3.08 seconds to compute the encryption for the same sized block. Micro-ECC [87] is an ECC implementation meant for microcontrollers[88].

Table 6.1: Average operation times for 100 runs[4].

	NISTP256	**RSA**
Time to generate key pair	0.52s	0.472s
Time to generate signature (32 bytes) packet	0.56s	0.908s
Time to encrypt (32 bytes) packet	N/A	0.308s

6.2 Design for secure reconfigurable power converters

Distributed Energy Resources (DER) such as solar photovoltaic (PV) systems, wind power systems, battery energy storage systems (BESS) utilize power converters such as dc-ac inverters and dc-dc converters to interface with the power grid. These converters use power electronic semiconductor devices, gate drives, sensors, and digital controllers. Functionally, the PV inverters process dc available power from the PV arrays and supply the real power to the grid. The battery inverters process bidirec-

tional power, enabling the charging and discharging of the batteries. Reconfigurable architectures such as Field Programmable Gate Arrays (FPGA) allow for finer control of power converters than a processor-based system. Logic on the FPGAs can be programmed to react to events at a higher speed than the processor-based system [89].

FPGAs offer signal processing capabilities in the form of Digital Signal Processing (DSP) blocks that are used in conditioning input /output signals[90]. FPGAs operate on concurrent logic as opposed to microprocessor-based systems which, if effectively employed can provide higher efficient control [91]. Additionally, the logic design on the SRAM based FPGAs can be reconfigured on runtime. This reconfigurability allows FPGAs based in-field device to be updated with the changing requirements. The open source and reconfigurable hardware extend the service life of the device.

Based on [4], [8] proposes a design for secure FPGA based power converters. A TPM based secure communication and certification framework is proposed in the design to mitigate attacks that may corrupt the communication between the backend SCADA system and a reconfigurable power converter. An attacker having access to the smart grid network can perform any number of remote attacks on a power converter: (1) An attacker may pose as the SCADA server to a connected power converter and send erroneous command inputs to the power converter, affecting its stability; (2) a third-party power converter joining the grid network may eavesdrop on the network communication or even attempt attacks on other connected nodes; (3) FPGA based power converters allow reconfiguration of hardware as well as software. For updating the system, the backend can also send hardware (bitstream) or software (firmware) updates. A malicious entity posing as a server can push tainted software to a power converter that may pose a threat either to the power converter or to the connected network.

Any device, first or third party, before is added to the grid must first be enrolled.

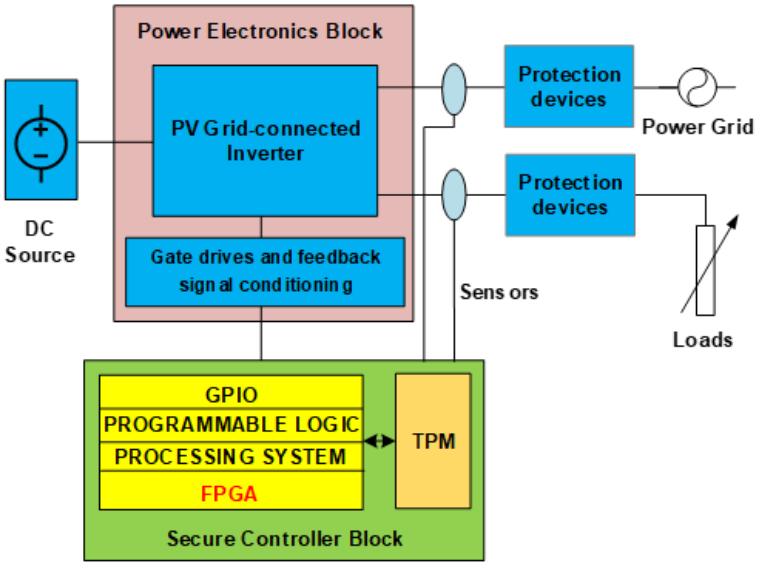

Figure 6.11: Reconfigurable secure power electronic converter framework[8].

The enrollment process is performed in a trusted environment. This can be a utility office for the grid. Identity of each power converter is established using asymmetric keys. Lightweight key provisioning schemes e.g. Elliptic Curve Digital Signature Algorithm (ECDSA) and other schemes are evaluated for the framework. ECDSA scheme uses a pair of public and private key to establish identity. The key provisioning scheme includes enrollment and regeneration for identification and authentication. During the enrollment process, the onboard TPM is used to first generate a primary key. This primary is used as a root key for the further key generation processes. TPM 2.0 offers a structure called TPM2_CREATE_PRIMARY.

This structure creates a primary key and returns the context for the key to the host machine. The key itself does not leave the TPM. Based on this key, using the TPM's TPM2_CREATE structure, set of a private and public key is generated. The private key is stored in the TPM's persistent tamper resistant memory using NV_WRITE structure. The public key is stored at the Registration Server(RS) for infield reference and authentication. Additionally, the server public key is stored on the node's TPM's memory. Enrollment ensures that no unauthorized node can become a part of the

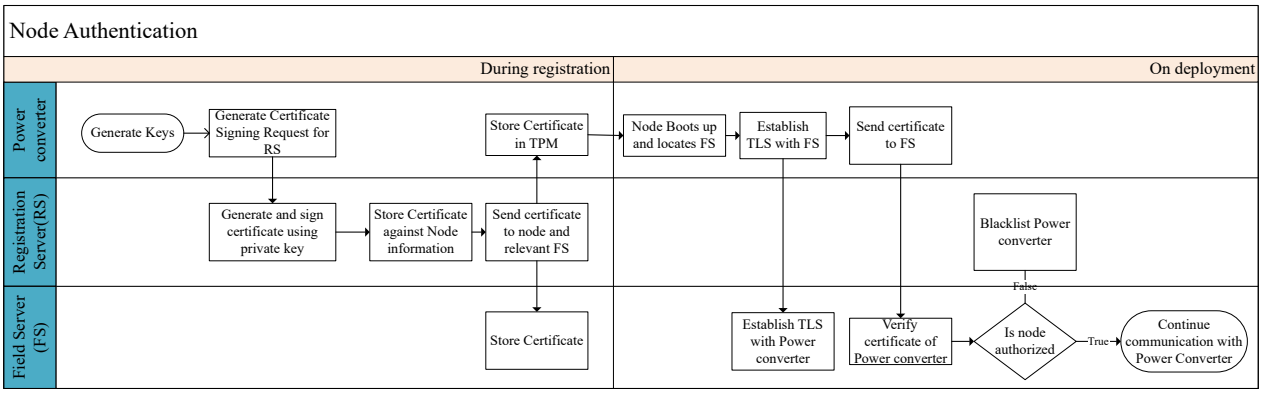

Figure 6.12: Node Authorization Scheme for Power Converters[8].

network. Required connection settings are also transferred to the client node. This may include the configuration that may be required for a client node to connect with the grid server.

Once a power converter has been enrolled with the RS, it can now be brought online as a part of the grid network. Using the network and server configuration passed to the client during the enrollment phase, the client attempts to connect with the grid. A client locates its assigned in-field server (FS) on the grid network. Before deployment, the RS sends every FS, the certificates of the end nodes FS expect communication with, on the field. Once FS is first located by an end node, the end node establishes a TLS session with the FS. For encryption, an end node uses TPM to create ephemeral encryption keys. TPM 2.0 specification supports ECDH based ephemeral key creation support using the command TPM2_EC_Ephemeral to create a pair of ephemeral keys and TPM2_ECDH_KeyGen to generate a symmetric encryption key from the node's private key and the public key of the FS.

On symmetric key generation, the TPM's AES encryption engine is now used for encrypting and decrypting session data. Once an encrypted session has been established, both the nodes exchange their certificates. The field server now authenticates the connecting power converter node by verifying the RS signed certificate. If the identity fails to verify, the field Server blacklists the offending node by sending a

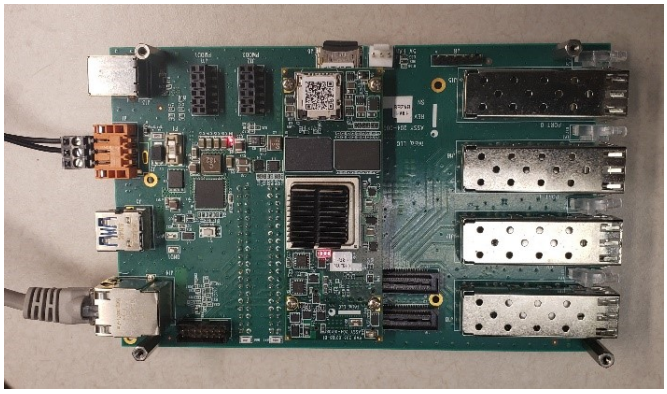

Figure 6.13: Iviea Atlas-I-Z8 board for Power Converter [8].

request to the registration server. This scheme is illustrated in Figure 6.13.

The presented architecture in [8] has been implemented on an Iviea Atlas-I-Z8 FPGA board [92] with Blackwing Carrier board. This board is equipped with Xilinx UltraScale+ ZU2CG FPGA core which has a dedicated PS and a PL region. The Controller Block design for the power converter is a combination of code implementation on the Processing System and logic implementation on the PL logic. The design is using a vendor provided Linux distribution with Kernel version 4.9. Power electronics control logic is implemented on the PL. Control logic implements a Pulse Width Modulation (PWM) which can control a DC to AC inverter. The design uses Infineon TPM 2.0 SLB9670 as a security platform, which is connected to the FPGA board using SPI. IBM's TPM Software Stack (TSS) is used to build tools required for accomplishing our framework.

6.3 Security Analysis

The TPM based secure framework provides secure key provisioning, secure boot process for smart grid devices, authentication and secure communication. The keys are generated and stored on a node's TPM in a trusted and a tamper-resistant environment, where the primary private key never leaves the node, and ephemeral keys are used for secure communication sessions. In case an attacker gets unauthorized access to the node, the private key stays safe and inaccessible. All the updates to the

keys are made in a trusted environment and/or by a trusted entity.

The secure communication halts the man in the middle and data spoofing is mitigated by the data signatures using the one-way hash functions. The secure boot guarantees the system is booted into a known state. To provide runtime guarantee, the FSBL can be extended to implement a watchdog-based interrupt to timely verify the integrity of the code that is executed on the system. The CA is responsible for signing the keys and the messages are signed using digital signatures. The client nodes periodically query the CA for an update, to access the new enrolled public keys of the area nodes. In case an update is available, it can simply be overwritten over the existing application package to add/update nodes.

6.4 Conclusion

This work demonstrates the state of security of the current power grid. The power grid is composed of edge devices that are resource constrained. The communication needs are and the requirement for security in such resource-constrained devices is presented. Based on the requirements, a secure communication framework is presented that uses modern security constructs such as TPMs, while also performing the necessary power processing and grid support functionality. The proposed secure framework can be extended beyond power electronic converters, to critical power components such as breakers, relays, and other distribution grid components.

CHAPTER 7: Conclusions and Future Work

Advancement in reconfigurable architectures, Internet of Things (IoT) and resource constrained devices have increased the complexity of these devices. This has lead to increased reliance on such devices for more tasks. Based on the application area, vendors require that their application code and hardware design is unchanged once the device is placed on the field. As existing works presented in this research have been shown to be inadequate for current trends, this work mitigates their shortcomings by presenting innovative schemes and frameworks.

Secure boot schemes for the bitstream ensure that any bitstream that is loaded onto the FPGA can first be verified for security that it came from a valid and integrity can be verified. The scheme presented in Chapter 3 builds on the limitation of the current solutions. Existing works implement security as a partition on the application bitstream. This approach firstly limits the area for application logic, secondly opens a door for malicious application logic IP to access sensitive data from the security partition. The solution presented in this research instead relies on well-established hardware based cryptographic solutions such as TPMs to provide a root of trust and an isolated secure environment.

Considering future work for this research, one direction is extending the role of security into runtime domain by check-pointing output responses generated from the application logic. A secure process thread should routinely be able to read the state of the circuit, a cumulative hash can be computed which can become one of the inputs in the key update scheme.

Bitstream based logic obfuscation scheme presented in Chapter 4 is a novel medium of providing runtime bitstream security. Existing security methods have considered

the bitstream as a monolithic block of data. However, this research has opened an avenue to implement intra-bitstream security. To apply bitstream obfuscation, no knowledge of the application is required and only the generated bitstream is used. This aids system designers that want a generic solution to provide runtime security. However, as shown, to improve runtime application security a logic locking scheme is also introduced. This provides an additional design based security.

A potential future research direction is using RTL design in the generation of bitstream obfuscated bitstream. Markers within the RTL code can provide metadata to the bitstream obfuscation process. These markers can indicate which LUT information should be corrupted and which ones should be blanked out. Additionally, the markers can also be used to exclude design elements from the obfuscation process.

REFERENCES

[1] Xilinx Inc., "UG 470 - 7 Series FPGAs Configuration." [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[2] A. S. Siddiqui, Y. G. J. Plusquellic, and F. Saqib, "Secure communication over CANBus," in *Midwest Symposium on Circuits and Systems*, vol. 2017-Augus. IEEE, aug 2017, pp. 1264–1267.

[3] A. S. Siddiqui, C.-C. Lee, W. Che, J. Plusquellic, and F. Saqib, "Secure Intra-Vehicular Communication over CANFD," in *IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, Beijing, China, 2017.

[4] A. S. Siddiqui, Y. Gui, D. Lawrence, S. Laval, J. Plusquellic, M. Manjrekar, B. Chowdhury, and F. Saqib, "Hardware assisted security architecture for smart grid," in *Proceedings: IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, oct 2018, pp. 2890–2895.

[5] A. S. Siddiqui, G. N. Shirley, S. R. Joseph, Y. Gui, J. Plusquellic, M. V. Dijk, and F. Saqib, "Multilayer camouflaged secure boot for SoCs," in *Microprocessor/SoC Test, Security & Verification (MTV19)*. IEEE, 2019.

[6] W. Che, F. Saqib, and J. Plusquellic, "PUF-based authentication," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, nov 2015, pp. 337–344.

[7] Y. Gui, A. S. Siddiqui, and F. Saqib, "Hardware Based Root of Trust for Electronic Control Units," in *Conference Proceedings - IEEE SOUTHEASTCON*, vol. 2018-April. IEEE, apr 2018, pp. 1–7.

[8] A. S. Siddiqui, P. R. Chowdhury, Y. Gui, M. Manjrekar, S. Essakiappan, and F. Saqib, "Design of Secure Reconfigurable Power Converters," in *2019 IEEE CyberPELS, CyberPELS 2019*, Knoxville, TN, 2019.

[9] K. Lasse Lueth, "State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating," 2018. [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[10] "Meet FPGA: The Tiny, Powerful, Hackable Bit of ..." [Online]. Available: https://www.darkreading.com/edge/theedge/meet-fpga-the-tiny-powerful-hackable-bit-of-silicon-at-the-heart-of-iot/b/d-id/1335730

[11] Wise Guy Reports, "Global SRAM FPGA Market Report 2019 - Market Size, Share, Price, Trend and Forecast- WiseGuyReports." [Online]. Available: https://www.wiseguyreports.com/reports/4144914-global-sram-fpga-market-report-2019-market-size-share-price-trend-and-forecast

[12] K. Ashton, "That 'Internet of Things' Thing," *RFiD Journal*, 2009. [Online]. Available: http://www.itrco.jp/libraries/RFIDjournal-ThatInternetofThingsThing.pdf

[13] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-Based Big Data Storage Systems in Cloud Computing: Perspectives and Challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2017.

[14] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[15] É. Morin, M. Maman, R. Guizzetti, and A. Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," *IEEE Access*, vol. 5, pp. 7097–7117, 2017.

[16] T. A. Youssef, M. E. Hariri, N. Bugay, and O. A. Mohammed, "IEC 61850: Technology standards and cyber-threats," in *EEEIC 2016 - International Conference on Environment and Electrical Engineering*. IEEE, jun 2016, pp. 1–6.

[17] R. Isermann, R. Schwarz, and S. Stölzl, "Fault-tolerant drive-by-wire systems," *IEEE Control Systems Magazine*, vol. 22, no. 5, pp. 64–81, oct 2002.

[18] O. Brandl, "V2X traffic management," *Elektrotechnik und Informationstechnik*, vol. 133, no. 7, pp. 353–355, nov 2016.

[19] A. Greenberg, "Hackers Gain Direct Access to US Power Grid Controls," *Wired*, pp. 1–8, 2017. [Online]. Available: https://www.wired.com/story/hackers-gain-switch-flipping-access-to-us-power-systems/

[20] T. Brewster, "Marriott Hackers Stole Data On 500 Million Guests - Passports And Credit Card Info Included," *Forbes*, pp. 19–22, 2018. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2018/11/30/marriott-admits-hackers-stole-data-on-500-million-guests/

[21] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks," *Usenix*, p. 5, 1998.

[22] M. Prandini and M. Ramilli, "Return-oriented programming," pp. 84–87, nov 2012.

[23] Xilinx, "Zynq-7000 All Programmable SoC Software Developers Guide," Tech. Rep., 2013. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf

[24] Xilinx Inc., "Zynq-7000 All Programmable SoC Secure Boot," 2014. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1025-zynq-secure-boot-gsg.pdf

[25] Y. Gui, S. Mohan Tamore, A. S. Siddiqui, and F. Saqib, "Key Update Countermeasure for Correlation-Based Side-Channel Attacks," *Journal of Hardware and Systems Security*, 2020.

[26] C. Herder, M. D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," pp. 1126–1141, aug 2014.

[27] A. Seshadri, A. Perrig, L. Van Doom, and P. Khosla, "SWATT: SoftWare-based ATTestation for embedded devices," in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2004. IEEE, 2004, pp. 272–282.

[28] "TPM Library Specification." [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_library_specification

[29] D. R. Engler, M. F. Kaashoek, and J. W. O'Toole, "The operating system kernel as a secure programmable machine," *ACM SIGOPS Operating Systems Review*, vol. 29, no. 1, pp. 78–82, 1995.

[30] W. Arbaugh, D. Farber, and J. Smith, "A secure and reliable bootstrap architecture," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*. IEEE Comput. Soc. Press, 1997, pp. 65–71.

[31] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Tech. Rep., 2008. [Online]. Available: https://tools.ietf.org/html/rfc5280https://www.rfc-editor.org/info/rfc5280

[32] G. Kim and E. Spafford, "Experiences with tripwire: Using integrity checkers for intrusion detection," 1994.

[33] A. S. Siddiqui, C. C. Lee, and F. Saqib, "Hardware based protection against malwares by PUF based access control mechanism," in *Midwest Symposium on Circuits and Systems*, vol. 2017-Augus, 2017, pp. 1312–1315.

[34] "Trusted Boot - Gentoo Wiki." [Online]. Available: https://wiki.gentoo.org/wiki/Trusted_Boot

[35] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN," in *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*. New York, New York, USA: ACM Press, 2015, pp. 1–6.

[36] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: a security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*. New York, New York, USA: ACM Press, 2014, pp. 1–14.

[37] T. Abera, N. Asokan, L. Davi, J. E. Ekberg, T. Nyman, A. Paverd, A. R. Sadeghi, and G. Tsudik, "C-FLAT: Control-flow attestation for embedded systems software," in *Proceedings of the ACM Conference on Computer and Communications*

Security, vol. 24-28-Octo. New York, New York, USA: ACM Press, 2016, pp. 743–754.

[38] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters," in *2015 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015*. IEEE, nov 2016, pp. 544–551.

[39] NXP, "Secure Boot on i.MX 50, i.MX 53, i.MX 6 and i.MX 7 Series using HABv4," 2018. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN4581.pdf

[40] Xilinx Inc., "Zynq UltraScale+ Device Technical Reference Manual UG1085," 2018. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf

[41] "Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1323-zynq-usp-tamper-resistant-designs.pdf

[42] D. Owen Jr., D. Heeger, C. Chan, W. Che, F. Saqib, M. Areno, and J. Plusquellic, "An Autonomous, Self-Authenticating, and Self-Contained Secure Boot Process for Field-Programmable Gate Arrays," *Cryptography*, vol. 2, no. 3, p. 15, jul 2018.

[43] I. Lebedev, K. Hogan, and S. Devadas, "Invited paper: Secure boot and remote attestation in the sanctum processor," in *Proceedings - IEEE Computer Security Foundations Symposium*, vol. 2018-July. IEEE, jul 2018, pp. 46–60.

[44] Xilinx and Inc, "Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream Application Note (XAPP1267)," 2017. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1267-encryp-efuse-program.pdf

[45] A. Carelli, C. A. Cristofanini, A. Vallero, C. Basile, P. Prinetto, and S. Di Carlo, "Securing bitstream integrity, confidentiality and authenticity in reconfigurable mobile heterogeneous systems," in *2018 IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2018 - THETA 21st Edition, Proceedings*. IEEE, may 2018, pp. 1–6.

[46] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, and G. Sigl, "How to break secure boot on FPGA SoCs through malicious hardware," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10529 LNCS. Springer, Cham, 2017, pp. 425–442.

[47] J. A. Roy, F. Koushanfar, I. L. Markov, J. Roy A., and I. Markov L., "EPIC: Ending Piracy of Integrated Circuits," *2008 Design, Automation and Test in Europe*, vol. 43, no. 10, pp. 1069–1074, 2008.

[48] S. M. Plaza and I. L. Markov, "Protecting integrated circuits from piracy with test-aware logic locking," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, vol. 2015-Janua, no. January. IEEE, 2015, pp. 262–269.

[49] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proceedings of the 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015*. IEEE, may 2015, pp. 137–143.

[50] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016*. IEEE, may 2016, pp. 236–241.

[51] Y. Xie and A. Srivastava, "Mitigating SAT attack on logic locking," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9813 LNCS. Springer, Berlin, Heidelberg, 2016, pp. 127–146.

[52] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proceedings of the 2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017*. IEEE, may 2017, pp. 95–100.

[53] F. Yang, M. Tang, and O. Sinanoglu, "Stripped Functionality Logic Locking with Hamming Distance-Based Restore Unit (SFLL-HD)-Unlocked," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2778–2786, oct 2019.

[54] S. Schmidt, M. Tausig, M. Hudler, and G. Simhandl, "Secure Firmware Update Over the Air in the Internet of Things Focusing on Flexibility and Feasibility," in *Internet of Things Software Update Workshop (IoTSU), At Dublin*, no. June, 2016.

[55] H. Chandra, E. Anggadjaja, P. S. Wijaya, and E. Gunawan, "Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development," in *Proceedings - Asia-Pacific Conference on Communications, APCC 2016*, 2016, pp. 115–118.

[56] C. E. Andrade, S. D. Byers, V. Gopalakrishnan, E. Halepovic, M. Majmundar, D. J. Poole, L. K. Tran, and C. T. Volinsky, "Managing massive firmware-over-the-air updates for connected cars in cellular networks," in *CarSys 2017 - Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services, co-located with MobiCom 2017*. New York, New York, USA: ACM Press, 2017, pp. 65–72.

[57] V. Zimmer and M. Krau, "Establishing the root of trust," 2016. [Online]. Available: http://www.uefi.org/sites/default/files/resources/UEFIRoTwhitepaper_Final8816(003).pdf

[58] R. Wilkins and B. Richardson, "Uefi Secure Boot in Modern Computer Security Solutions," *UEFI Forum*, 2013. [Online]. Available: http://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[59] "Secure the Windows 10 boot process | Microsoft Docs." [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/secure-the-windows-10-boot-process

[60] J. Zhang, Y. Lin, and G. Qu, "Reconfigurable Binding against FPGA Replay Attacks," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 2, pp. 1–20, mar 2015.

[61] Xilinx, "AXI Hardware ICAP." [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_hwicap.html

[62] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Design and Test*, vol. 30, no. 2, pp. 45–54, apr 2013.

[63] NIST, "RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE," 1999.

[64] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to Remote Attestation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[65] "WebHome - U-Boot - DENX." [Online]. Available: https://www.denx.de/wiki/U-Boothttp://www.denx.de/wiki/U-Boot/WebHome

[66] AVNET, "ZedBoard | Zedboard." [Online]. Available: http://zedboard.org/product/zedboard

[67] "SLB 9670VQ2.0 - Infineon Technologies." [Online]. Available: https://www.infineon.com/cms/en/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slb-9670vq2.0/

[68] Xilinx and Inc, "Programming ARM TrustZone Architecture on the Xilinx Zynq7000 All Programmable SoC User Guide (UG1019)," 2014. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1019-zynq-trustzone.pdf

[69] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "SACHa: Self-Attestation of Configurable Hardware," in *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*. IEEE, mar 2019, pp. 746–751.

[70] M. Ender, M. Wilhelm, P. Swierczynski, P. M. Knopp, S. Wallat, and C. Paar, "Insights into the mind of a Trojan designer: The challenge to integrate a trojan into the bitstream," in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*. Institute of Electrical and Electronics Engineers Inc., jan 2019, pp. 112–119.

[71] S. Wallat, N. Albartus, S. Becker, M. Hoffmann, M. Ender, M. Fyrbiak, A. Drees, S. Maaen, and C. Paar, "Highway to HAL: Open-Sourcing the First Extendable Gate-Level Netlist Reverse Engineering Framework," in *ACM International Conference on Computing Frontiers 2019, CF 2019 - Proceedings*, 2019, pp. 392–397. [Online]. Available: https://github.com/emsec/hal

[72] SymbiFlow, "SymbiFlow - the GCC of FPGAs." [Online]. Available: https://symbiflow.github.io/

[73] "SymbiFlow/prjxray: Documenting the Xilinx 7-series bit-stream format." [Online]. Available: https://github.com/SymbiFlow/prjxray

[74] J. Aarestad, P. Ortiz, J. Plusquellic, and D. Acharyya, "HELP: A hardware-embedded delay PUF," *IEEE Design and Test*, vol. 30, no. 2, pp. 17–25, apr 2013.

[75] W. Che, M. Martin, G. Pocklassery, V. Kajuluri, F. Saqib, and J. Plusquellic, "A Privacy-Preserving, Mutual PUF-Based Authentication Protocol," *Cryptography*, vol. 1, no. 1, p. 3, nov 2016.

[76] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, aug 2001.

[77] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, 2015.

[78] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, "Poster: Hardware based security enhanced framework for automotives," in *2016 IEEE Vehicular Networking Conference (VNC)*. IEEE, dec 2016, pp. 1–2.

[79] W. Che, V. Kajuluri, M. Martin, F. Saqib, and J. Plusquellic, "Analysis of Entropy in a Hardware-Embedded Delay PUF," *Cryptography*, vol. 1, no. 1, p. 8, jun 2017.

[80] FICS, "2017 FICS Research Conference Poster Winners - Florida Institute for Cybersecurity Research - University of Florida," 2017. [Online]. Available: https://fics.institute.ufl.edu/2017-fics-research-conferenbce-poster-winners/?doing_wp_cron=1564510713.2381839752197265625000

[81] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, "A Secure Communication Framework for ECUs," *Advances in Science, Technology and Engineering*

Systems Journal, vol. 2, no. 3, pp. 1307–1313, aug 2017. [Online]. Available: http://astesj.com/v02/i03/p165/

[82] O. Blog, "A Look Back at SCADA Security in 2015 | OPSWAT Blog." [Online]. Available: https://www.opswat.com/blog/look-back-scada-security-2015

[83] H. Chae, A. Shahzad, M. Irfan, and H. Lee, "Industrial Control Systems Vulnerabilities and Security Issues and Future Enhancements," 2015, pp. 144–148.

[84] S. Biddle, "(Known) SCADA Attacks Over The Years," 2015. [Online]. Available: https://blog.fortinet.com/2015/02/12/known-scada-attacks-over-the-years

[85] Brian Krebs, "FBI: Smart Meter Hacks Likely to Spread - Krebs on Security," 2012. [Online]. Available: https://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/

[86] Q. A. Al-Haija, M. A. Tarayrah, H. Al-Qadeeb, and A. Al-Lwaimi, "A Tiny RSA Cryptosystem based on Arduino Microcontroller Useful for Small Scale Networks," *Procedia Computer Science*, vol. 34, pp. 639–646, jan 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914009466

[87] "micro-ecc." [Online]. Available: http://kmackay.ca/micro-ecc/

[88] M. Tausig and S. Schmidt, "Performance Evaluation of Cryptographic Operations on a SAMR21-XPRO Board," *researchgate.net*.

[89] A. Stumpf, D. Elton, J. Devlin, and H. Lovatt, "Benefits of an FPGA based SRM controller," in *Proceedings of the 2014 9th IEEE Conference on Industrial Electronics and Applications, ICIEA 2014*. IEEE, jun 2014, pp. 12–17.

[90] M. Dagbagi, L. Idkhajine, E. Monmasson, and I. Slama-Belkhodja, "FPGA implementation of Power Electronic Converter real-time model," in *SPEEDAM 2012 - 21st International Symposium on Power Electronics, Electrical Drives, Automation and Motion*. IEEE, jun 2012, pp. 658–663.

[91] A. De Castro, P. Zumel, O. García, T. Riesgo, and J. Uceda, "Concurrent and simple digital controller of an AC/DC converter with power factor correction based on an FPGA," *IEEE Transactions on Power Electronics*, vol. 18, no. 1 II, pp. 334–343, jan 2003.

[92] Iveia, "Atlas-I-Z8 Low-Power Zynq UltraScale+ SoM - iVeia." [Online]. Available: http://iveia.com/atlas-i-z8