A Dependable and Secure Approach for Secret Key Establishment and Operation in Automotive CPS

by

Naresh Kumar Giri

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Arslan Munir

Copyright

© Naresh Kumar Giri 2018.

Abstract

Modern automobiles incorporate a network of electronic control units (ECUs) that provides a range of features such as safety, driver assistance, infotainment. Such network of ECUs in a vehicle are connected to each other through buses, forming interconnections called intra-vehicle network. Bus technologies that are widely used in modern day automobiles are controller area network (CAN), local interconnect network (LIN), and media oriented systems transport (MOST). These bus technologies, however, do not possess any security or dependability features, and thus are susceptible to vulnerabilities. Such vulnerabilities allow attackers to mount passive attacks (e.g., snooping) and/or active attacks (e.g., fault injection). In this study, we propose a scheme for secure authentication of automotive ECUs. Our proposed scheme ensures that only authenticated ECUs can participate in communication over the intra-vehicle network/bus. ECU authentication is carried out using certificate-based authentication which is implemented using elliptic curve cryptography (ECC). The study also proposes a symmetric (session) key-establishment mechanism within intra-vehicular network to establish a common symmetric (session) key for all ECUs to communicate over the network. The key-establishment mechanism removes the need of storing symmetric keys in ECU memory permanently. The study incorporates key refreshment by assigning a certain lifetime/timeframe period to symmetric (session) key and then regularly updates session key after the expiration of each lifetime. Our proposed method provides confidentiality and integrity in intra-vehicle ECU communication without violating safety and real-time constraints of the vehicle. Our approach leverages multi-core ECUs to provide fault-tolerance by using redundant multi-threading (FT-RMT), performs quick error detection (FT-QED) and accelerate performance using lightweight checkpointing (CP).

Table of Contents

List of Figures

List of Tables

Acknowledgments

This work was supported by the National Science Foundation (NSF) (NSF-CNS-1743490). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

I would like to express my sincere gratitude towards my advisor Professor Arslan Munir for granting me the opportunity to be a part of his research group, ISCAAS. My academic training would be incomplete with the mentorship and generous support of Professor Munir.

I would also like to extend my gratitude towards my committee members Professor Mitchell Neilson and Professor Eugene Vasserman for their time and expert views and opinions on this study.

I would also like to take this opportunity to thank my colleagues Prasanna Kansakar and Bikash Poudel for constantly pushing me towards exciting new platforms, both academically and professionally. Thank you to Bibek Bhattarai and Avantika Gurung for providing me a helping hand whenever needed. I would also like to acknowledge my friends, seniors and relatives who were directly or indirectly involved.

Lastly, I am indebted to my mother, father and brother for providing me with emotional and moral support all the way from Nepal.

Dedication

To my parents.

Chapter 1

Introduction and Motivation

Advancements in computing, sensor, and communication technologies have transformed automotive systems from *dumb* electro-mechanical devices into *smart and intelligent*. Modern vehicles are equipped with a multitude of sensors, digital processors (known as Electronic Control Units (ECUs)), and radio interfaces connected to each other via in-vehicle bus networks and protocols. In addition, modern automobiles employ several communication protocols such as Controller Area Network (CAN) and CAN with flexible data rate (CAN FD) for carrying engine control, transmission control messages, Local Interconnect Network (LIN) for carrying headlamp, electric windows and doors signals, Media Oriented System Transport (MOST) for carrying infotainment systems signal, and Ethernet for video acquisition to support autonomous driving. FlexRay is also a recently introduced protocol for automotive x-by-wire applications.

Highly critical control messages between components such as engine control module (ECM), anti-lock braking system (ABS), and electric steering system (ESS) are sent over the bus via protocols like CAN, CAN-FD, and FlexRay depending upon underlying network technologies. However, these protocol do not have built-in security primitives. Attackers can easily gain access to any ECU through a compromised bus network and can read/alter messages potentially causing security threats. An attacker who has direct access to any ECU can control many safety critical systems such as disabling brakes, stopping the engine, opening doors, changing heating and cooling, and turning on/off lights [1], [2].

Previous works [3], [4] have incorporated security primitives like Advanced Encryption Standard (AES) along with Hash-based Message Authentication Code (HMAC) for message integrity. To meet real-time deadline requirements, these approaches use computationally economic symmetric cryptography. However, symmetric cryptography requires pre-shared encryption keys between communicating parties. Existing works assume that these keys are programmed by the original equipment manufacturers (OEMs) during vehicle manufacturing and assembly. OEMs tend to use identical keys across series of ECUs and even vehicles. This makes a whole series of ECUs and vehicles vulnerable to security attacks when a single key is compromised. Moreover, symmetric keys can be easily extracted using side-channel analysis (SCA) attacks. This renders use of permanent symmetric keys in all ECUs very vulnerable to attacks.

Although use of unique symmetric keys for every ECU can alleviate the risk, it increases system complexity significantly as every ECU in the intra-vehicle network needs to store symmetric keys for all other ECUs in the network. In addition, if any ECU of a vehicle is compromised through its storage, then an attacker may obtain the symmetric keys for all ECUs.

In this work we propose a *symmetric (session) key-establishment mechanism*, which generates, distributes and periodically updates the symmetric session keys for all ECUs to communicate via intra-vehicle network. While node-to-node communication is carried out using symmetric cryptography, to ensure that real time deadlines are always met, our solution eliminates the need to store the symmetric keys permanently. This prevents an attacker from gaining access to symmetric keys through permanent storage. The key refreshment policy prevents replay and side channel attacks. In addition, ECUs are authenticated using *certificate-based authentication* before being supplied with session keys. This before-hand authentication helps to prevent malicious ECUs from getting symmetric keys by infiltrating into the network.

Authentication and key distribution have been widely studied and protocols such as Kerberos protocol [5] has been developed. Kerberos is a key exchange protocol that utilizes pre-shared symmetric keys to generate symmetric session keys that expire after a certain lifetime period associated with the session. The key exchange in Kerberos is managed by a key distribution centre (KDC) which stores pre-shared symmetric keys in non-volatile memory for all the communicating nodes in the network. The pre-shared symmetric keys stored in non-volatile memory are vulnerable to SCAs [6]. An attacker can target the KDC and do SCAs to extract pre-shared keys of all ECUs in an in-vehicle network. Hence, this approach is not suitable for use in safety-critical automotive cyber-physical system (CPS). Instead, certificate-based asymmetric cryptography is more appropriate method for authentication and key exchange for automotive systems because this approach does not have to store all pre-shared keys.

For certification and authentication, we use elliptic curve cryptography (ECC). ECC is chosen over other asymmetric cryptography like RSA, Elgamal because ECC provides higher security with comparatively smaller key length [7]. ECC has 80-bit symmetric key security level with key length of 160 bits whereas for the same security level RSA requires a key of length of 1024 bits. ECC implementations therefore have lower computation complexity and are more suitable for applications having real-time deadlines.

For authentication, each ECU must ensure that all other ECUs are authentic which is an expensive task with computational complexity of $O(n^2)$. It also increases the network overhead as all nodes need to communicate with all the remaining nodes. Finally, an ECU that is responsible for generating and distributing key needs to be identified.

This problem of ECU authentication is addressed by introducing a Central Security Module (CSM) to do authentication of all ECUs. The CSM module authenticates all other ECUs by sharing its certificate and other ECUs authenticate with CSM by sharing their certificates. If any malicious ECU tries to authenticate with false certificate then CSM notifies through dashboard. These certificates are generated and managed by a certificate authority (CA) specified by OEM.

CSM module also generates symmetric keys and shares these keys with all ECUs. Along with the symmetric keys, HMAC of keys is also sent to prevent man-in-the-middle (MITM) attack. The CSM module will periodically update the symmetric keys during runtime, which makes the system safe against SCA attacks.

Next generation automotive require dependability and safety features embedded in ECUs and in-vehicle networks. These requirements are strictly specified in ISO 26262 [8]. The ISO 26262 standard requires that at least one critical fault must be tolerated by an automobile without loss of functionality. To address this requirement we have incorporated fault tolerance (FT) by redundant multi-threading (RMT). This FT approach leverages RMT on a dual-core architecture. RMT uses two different threads to compute the same safety-critical computation. The result of the two different threads are matched at the end of the computation to detect any error during computation. If an error occurs during computation, recomputation is carried out in both threads. Recomputation fixes the errors that are caused by transient faults which consists of majority of errors.

We further enhance RMT by quick error detection (QED) mechanism to accelerate the computation during transient faults [9]. We further propose the use of lightweight checkpointing to alleviate the overhead of the complete/whole program recomputation in presence of faults.

We analyze and study the result of different fault-tolerant mechanisms and test the validity of our approach using a steer-by-wire (SBW) as case study.

In summary, our main contributions are as follows:

- Proposal of an integrated dependability and security approach that simultaneously incorporates security (key establishment, confidentiality, integrity, and authentication) and fault tolerance primitives while adhering to real-time constraints of automotive CPS. We demonstrate this approach through a steer-by-wire case study.

- Proposal of a novel certificate-based authentication scheme of ECUs leveraging ECC to prevent participation of unauthorized ECUs in in-vehicle network communication.

- Proposal of a symmetric (session) key-establishment protocol for ECUs in intra-vehicle networks to enable the ECUs to communicate securely over the in-vehicle networks.

- Integration of dependability primitives through various fault tolerance (FT) approaches such as FT by redundant multi-threading (FT-RMT), FT-RMT enhanced with quick error detection (FT-RMT-QED), and lightweight checkpointing.

The rest of the paper is organized as follows. Chapter 2 provides overview of existing solutions. Chapter 3 describes the preliminaries and underlying concepts of in-vehicle network architecture. Chapter 4 details the techniques and implementation mechanisms used in this work. Chapter 5 discuss the steer-by-wire model which is used as case study to verify our approach. Chapter 6 presents experimental results and analysis. Finally, Chapter 7 presents concluding remarks.

Chapter 2

Related Works

Many previous works have studied security of automotive systems.

Koscher et al. [1] analyzed internal and external attack surfaces through which an attacker could control automotive subsystems. In [1], authors attacked a car through onboard diagnostics (OBD) port by using a self-developed software, where authors were successfully controlled radio, instrument panel cluster, body controller, engine, brakes and heating ventilation air conditioning (HVAC). Rouf et. al. [10] have studied the security and privacy related to Tire Pressure Monitoring System (TPMS). Huang et al. [11] classified the in-vehicle network in three different layers i.e. control layer, middle layer and external interface layer, and studied the security vulnerabilities at each layer. All these works suggested that there needs to be security and authentication mechanisms within in-vehicle networks for realizing secure and dependable automotive CPS.

Lin et al. [12] proposed integration of message authentication codes (MACs) in CAN data frames to prevent masquerade and replay attacks. Wolf et al. [13] proposed a vehicular hardware security module (HSM) that provided hardware support for symmetric cryptography, asymmetric cryptography, hash function, and pseudorandom number generator. However, the HSM did not support any FT features which are crucial for safe operation of modern automobiles. The dependability for automotive embedded system has been studied in some previous works. Beckschulaze et. al.[14] have studied different FT approaches on dual-core micro-controllers. Munir et al. [3], Poudel et al. [4] have proposed multicore ECU based design for secure and dependable cybercars. However, these works did not discuss about establishment and sharing symmetric secret key.

Chapter 3

Background

In this chapter, we discuss about in-vehicle functional domain and in-vehicle network. We also discuss about Elliptic curve cryptography (ECC) which is used in proposed protocol.

3.1 In-Vehicle Functional Domains

Early vehicles used dedicated point-to-point connections for in-vehicle communications. As the number of in-vehicle features increased, the point-to-point communication started becoming incompatible as the wiring system became bulkier, more complex, and difficult to install and manage. The introduction of serial bus addressed the issue of incompatible point-to-point connections in in-vehicle communications, through the reduction of wiring complexity.

Data rates in automobiles vary according to the different features; as engine modules and transmission modules require higher data than the data required for door and light module. Also, high-speed components are more expensive than low-speed components, and therefore, it is more cost effective to use high-speed components where high data rate is required and use low-speed components where low data rate is required. High speed buses are generally used for powertrain and engine control, and low speed buses are used for body electronics. For certain operations, the data between high-speed bus and low-speed bus has to be exchanged, and the exchange of data is done by a gateway device. When required to transfer messages, the gateway device converts messages from one protocol to other.

In this section we discuss about various functional domains in car embedded system.

3.1.1 Powertrain Domain

The powertrain domain is one of the two functional domains which are geared particularly for safety of the vehicle and real time control. The powertrain domain consists of two processes - i.) engine control-the power generation in the engine, ii.) transmission and gear control-power transmission in the engine through the gearbox to the wheels and driving axis. As the powertrain domain functions to control and manage the engine, it requires high computing power and complex control laws. This domain is formed through real-time subsystems that are frequently interacting and exchanging data with the body domain (e.g., dashboard) and the chassis (e.g., ABS). Therefore, the powertrain domain requires high dependability, communication predictability and high network bandwidth. Powertrain domain subsists high degree of network loads and message traffics and adapts with less flexibility.

3.1.2 Chassis Domain

The chassis domain is the other functional domain geared for safety of the vehicle and real time control. This chassis domain manages the functions of suspension, steering and braking and controls activity of driving dynamics, active safety and assistance. Such activities are monitored by systems such as active suspensions, electronic stability program (ESP), ABS, automatic stability control (ASC), adaptive cruise control (ACC), anti-slip regulation (ASR), electronic power steering (EPS), 4-wheel drive (4WD) and electronic damper control (EDC). This functional domain, similar to the powertrain domain, has advanced real-time control systems and closed loop with specific timing and safety applications.

Technologies such as x-by-wire are currently replacing mechanical and hydraulic systems to electrical systems which are being implemented in braking, driving and steering functions. Such innovative technologies fall in this domain. These functionalities also require high bandwidth, flexibility as well as high dependability.

3.1.3 Body Domain

The elements that make up the body domain are the body and comfort related functions such as air conditioning, wipers, doors, seats, climate control, locks, cruise control (CC), park distance control. Passenger activities such as locking doors and closing windows prompts the activation of the body domain. These elements of the body domain are controlled by electronic subsystems. As a variety of communications needs are found in body domain, the network architecture is formed in a hierarchical manner where subsystems are connected via the CAN backbone. From a safety standpoint, this domain is not considered safety critical nor does it require higher bandwidth. Low cost networks such as LIN is used to implement communication subsystem of this domain.

3.1.4 Telematics Domain

This domain can consist of vehicular technologies and telecommunication elements ranging from wireless network, in-vehicle navigation systems, CD/DVD players, rear seat entertainment, audio systems, monitors, displays, infotainment and multimedia. Wireless technologies in automobiles can provide services such as hands-free phones, laptop computers, GPS units, car access systems in the telematics domain. This domains wireless technology is not only limited to communication, but also other diverse functions such as traffic information systems, fleet management systems, voice recognition, diagnostics and maintenance services and many more.

The nature of this domain requires gigantic amount of data to be circulated and exchanged through systems as well as the external world. Therefore, extensibility (principle where implementation takes future growth into consideration) and composability (principle which deals with the inter-relationships of components) requirements are crucial for telematics domain, unlike embedded real-time networks in the chassis and powertrain domains. This domain is more focused on bandwidth sharing, data streams of multimedia, quality of service of multimedia while preserving and ensuring harmful alterations and confidentiality of information.

3.1.5 Passive Safety Domain

This domain serves particularly safety related functions, employing electronic based systems that serves to protect occupants in the vehicle. Prominent examples of such electronic based systems are belt pretensioners, tire monitoring, roll over sensors and airbags.

3.2 In-Vehicle Network

In this sub-section bus networks that are widely used in modern vehicles are discussed.

3.2.1 Local Interconnect Network (LIN)

LIN is a low cost and low speed (20 kbps) serial bus in-vehicle communication network. The application of this protocol are on areas that are neither time critical nor needing extra fault tolerance. This network is used in body domain. It connects modules like light module, window module, sun-roof module, seat control module, climate control module, door module, mirror module, and other comfort functional modules.

LIN is time-triggered network, means message transmission is driven at predefined time instant [15]. LIN is based on universal asynchronous receiver-transmitter (UART) networking architecture. It is a single master/multiple slave bus that uses a single wire to transmit data. The LIN bus frame contains two parts, first part is a message header and the second part is message response. The master node controls the communication between various slaves. The master node sends the message request by sending the message header on the bus, the particular slave node replies the request by sending message response.

The major drawback of this protocol is that if the master node fails the whole communication network fails. Hence, it is not suitable for safety critical applications.

3.2.2 Media Oriented System Transport (MOST)

MOST is a protocol for high-speed multimedia and infotainment networking for automotive. This protocol provides an efficient and cost-effective audio, video, data and control information between any two nodes. MOST corporation, a consortium of car makers, system architects and key component suppliers was established in 1998. MOST protocol is adopted by every car brand.

The MOST protocol follows the seven layers of Open System Interconnection (OSI) model of communication protocol. MOST network is usually arranged in a ring topology, however, it can also be laid as star topology. This protocol can accommodate at most 64 nodes at a time. Plug and play functionality of this protocol makes it easier to add and remove any node from the network.

MOST is a synchronous network and it uses point-to-point data transfer, supporting both synchronous and asynchronous traffic. In MOST network, one device is designated as the timing master and rest all are slaves. Timing master continuously feeds the MOST frames into the ring. The preamble is sent out by timing master at beginning of transmission of MOST frame. The timing followers (slaves) use this preamble for synchronization.

3.2.3 Ethernet

Ethernet is one of the most common high-speed interfaces found in homes and offices. The interest from car manufacturer (e.g. BMW, Daimler), automotive electronics companies (e.g. Bosch, Continental, Micrel), and academic research projects are investigating the performance of Ethernet/IP in automotive embedded systems. The major advantage of Ethernet over traditional automotive bus technology is the bandwidth provided by Ethernet. Old technologies like LIN, MOST, CAN were tailored for automotive applications of that time. But recent development in automotive industry needs larger bandwidth technologies.

Ethernet is preferred for data-intensive requirement like driver assistance, collision avoidance, lane departure detection, traffic sign classification, blind spot detection, driver intent detection, pedestrian detection, and many others.

3.2.4 CAN

CAN bus is most widely used in-vehicle network that was developed by Robert Bosh GmbH in 1985. It became ISO standard in 1994. Although, it was originally designed for automotive industry but it is also popular in other fields like electric power, petroleum, chemical, metallurgical, aviation, industrial, and security protection. It is famous for its low-cost, its robustness, and the bounded communication delays. CAN provides the serial-communication upto 1Mbps and permits, 40 meter of bus-range [16]. Fault-tolerant systems use this bus because of its error detection and signaling mechanism. In case of any error in CAN frame, the ECUs either discards the frame, or ask for re-transmission the frame or raises error flags.

The standard CAN 2.0 frame structure had defined two frame structures CAN 2.0A and CAN 2.0B. CAN 2.0A structure consists of 7 fields: start of frame (SOF) bit, 18 bits header, 0-8 bytes of data field, 15 bit cyclic redundancy check (CRC) field, 3 bit acknowledgement field (ACK), 7 bits end of frame (EOF). The header of the frame consists of 11 bits identifier field, 1 bit of remote transmission request (RTR) field, 2 reserved bits and 4 bits data length code (DLC). The CAN 2.0B format is same as CAN 2.0A format except the identifier field of header is 29 bits instead of 11 bits. This identifier part defines the message priorities.

3.2.5 CAN-FD

CAN-FD is newer version of CAN network, that provides higher bandwidth with maximum bit-rate of 15Mbps. The protocol is backward compatible , so can work with CAN protocol. This protocol can communicate with data length of 8-64 bytes at a time.

3.2.6 FlexRay

FlexRay protocol was developed by FlexRay consortium of big automobile companies whose members were BMW, Bosch, DaimlerChrysler, General Motors, Motorola, Philips and Volkswagen. The aim of the consortium was to conduct technical analysis of existing networks which could meet all the technical requirement of modern automobiles. Since, most of available in-network protocols were found to be technically insufficient for newer automobiles, they came up with their own new protocol "FlexRay protocol". FlexRay provides high speed, fault-tolerant and flexible in-vehicle network communication. FlexRay can offer speed of upto 10Mbps as bus, star and multiple star networks. The bus operation time cycle is divided into two parts: the static segment and the dynamic segment. The static segment is time-triggered traffic in which particular slots are allocated for specific nodes. The dynamic segment is event-triggered traffic in which nodes take control of the bus according message identifiers.

FlexRay supports dual channel which provides redundancy for dependability and higher bandwidth communication. This also supports CRC, bus guardians, never-give-up strategy (strategy of nodes to get into safe mode after transient fault) of nodes and trigger monitoring mechanisms which makes it more suitable for dependable for safety-critical applications. However, implementation of FlexRay protocol is challenging and more expensive than contemporary networks. Furthermore, since this protocol is new compared to other protocols, only few automobile contain this protocol.

All the bus networks that are widely used in in-vehicle networks do not have security mechanisms embedded within them. CAN network, which is present in every vehicle used to carry control signals, does not have any built-in security primitive. Since, majority of the vehicles use CAN bus and most existing attacks on vehicle attacks are based on CAN network. CAN network is considered for symmetric key generation and distribution for secured communication of ECU nodes, however, our approach can be extended to other vehicle networks as well.

3.3 Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) is the newest member of the public-key algorithm. ECC provides the same level of security as RSA or discrete logarithmic (DL) systems with considerably shorter operands or key length. It is based on elliptic curve defined over a finite field (Galois Field (GF)). In finite field operation, all the arithmetic is performed modulo a prime ‘p’.

According to [7], elliptic curve ε is defined as the curve over $\mathbb{Z}_p$, where, $\mathbb{Z}_p$ is finite field and p is prime number $p > 3$, is the set of all pairs $(x, y) \in \mathbb{Z}_p$ which satisfies

$$\varepsilon(x, y) : y^2 \equiv x^3 + a.x + b \; mod \; p \tag{3.1}$$

together with an imaginary point of infinity ϕ, where, $a, b \in \mathbb{Z}_p$ and the condition $4 \cdot a^3 + 27 \cdot b^2 \neq 0 \; mod \; p$. Point of infinity ϕ is the neutral element required for finite field.

The operation on elliptic curves are called as group operations. The two main group operations point addition and point doubling. Given two points in elliptic curve, say $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ such that $P \neq Q$, **point addition** computes the third point $R = (x_3, y_3)$ such that

$$P + Q = R \tag{3.2}$$

and **point doubling** computes third point R when $P = Q$, given as

$$P + P = R \tag{3.3}$$

The value for third point R is computed by following equation,

$$\begin{aligned} x_3 &= s^2 - x_1 x_2 \; mod \; p \\ y_3 &= s(x_1 - x_3) - y_1 \; mod \; p \end{aligned} \tag{3.4}$$

where ,

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \; mod \; p; \; if \;\; P \neq Q \text{ (point addition)} \\ \frac{3x^2 + a}{2y_1} \; mod \; p; \; if \;\; P = Q \text{ (point doubling)} \end{cases}$$

The total number of points in a curve is finite denoted by $\#E$ and is called order of elliptic curve.

3.3.1 Public and Private Key Generation

Algorithm 1: ECC Public and Private Key Generation Algorithm

```
Input: Elliptic curve $\varepsilon$ ,
generation point $G = (G_x, G_y)$,
prime number $p$,
private key $(k_{pr} = k)$
Output: public key $k_{pub}$
$k_{pr} = random\ number\ k\ \in (1, \#E)$
$k_{pub} = k_{pr} \cdot G = k \cdot G$
```

The user's public key is a point in the curve and its private key is a randomly selected number. The base point or generation point G of the curve is multiplied with the private key to obtain public key. The basic algorithm for ECC private and public key generation is given in Algorithm 1. At first a random number, as private key k, less than order of curve $\#E$ is selected. The private key is then multiplied with generation point G to obtain the public key. This multiplication operation is also called point multiplication.

3.3.2 Scalar Multiplication (Point Multiplication)

Algorithm 2: Double And Add Algorithm

```
Input: Elliptic curve $\varepsilon$
generation point $G = (G_x, G_y)$,
prime number $p$
scalar $k = \sum_{i=0}^{t} k_i 2^i$ with $k_i\ in\ 0, 1$ and $k_t = 1$
such that $1 \leq k \leq q$ where $q = (\#E - 1)$
Output: $T = kG$
T = G
For $i = t - 1$ downto 0
  i) $T = T + T\ mod\ p$
  ii) If $k_i = 1$
        $T = T + G\ mod\ p$
Return $(T)$
```

Point multiplication operation involves many point addition and point doubling. Various algorithm are developed to quicken this operation. Double and Add Algorithm is one of the algorithm point multiplication operation. The basic steps for Double and Add Algorithm are shown in Algorithm 2. The scaler number k is expressed in binary by t number of bits. The algorithm iterates through the bit values from left to right; in each iteration it performs point doubling and only if the current bit value is 1 it performs point addition of G.

3.3.3 Elliptic Curve Diffie-Hellman key sharing (ECDH) Algorithm

Another important functionality of public key cryptography is addressing key distribution problem. ECC also provides mechanism for sharing a common secret communicating over an insecure channel. This key sharing mechanism is called elliptic curve Diffie-Hellman key exchange (ECDH). Before participating in key sharing mechanism two parties agree on domain parameters like elliptic curve (ε), prime number (p), generation point (G). The basic steps for ECDH is given in Algorithm 3. Here, side A and side B are sharing a common secret. Side A sends it's public key ($k_{pub,A}$) to side B. Since public keys are known to everyone losing this information will not compromise security system. At side B, it computes a common secret (T_{AB}) multiplying private key of B ($k_{pr,B} = b$) and public key of A($k_{pub,A}$). Then side B sends it's public key to side A. Now, side A also computes the common secret (T_{AB}) by doing point multiplication of private key of A ($k_{pr,A} = a$) with public key of B($k_{pub,B}$). As shown in step 2 and 4, the common secret value ($T_{AB} = a \cdot b \cdot G$) is shared in between two parties without losing any secured information.

Algorithm 3: Elliptic Curve Diffie-Hellman Key Exchange (ECDH) Algorithm

```
Input: Elliptic curve $\varepsilon$,
generation point $(G_x, G_y)$,
private key of A $(k_{pr,A} = a \in (1, \#E))$,
public key of A $(k_{pub,A} = a \cdot G)$,
private key of B $(k_{pr,B} = b \in (1, \#E))$,
public key of B $(k_{pub,B} = b \cdot G)$
Output: Common Secret $T_{AB} = (x_{AB}, y_{AB})$
A sends public key $k_{pub,A}$ to B
B computes $T_{AB} = k_{pr,B} \cdot k_{pub,A} = b \cdot a \cdot G$
B send public key $k_{pub,B}$ to A
A computes $T_{AB} = k_{pr,A} \cdot k_{pub,B} = a \cdot b \cdot G$
```

Chapter 4

Integrated Dependable and Secure Approach (IDSA)

The design of dependable and secure automotive CPS requires inclusion of dependability and security primitives without violating real-time constraints using limited resources (in terms of computation). This requirement has made the task of designing dependable and secure automotive CPS challenging.

This work includes different mechanisms for FT like FT-RMT, FT-RMT-QED and FT-RMT-CP which are explained in section 4.1. For secure communication between in-network vehicle nodes, security primitives proposed in [3], [4] are encryption for message confidentiality and hash based message authentication for message integrity. Both of these operations need symmetric keys shared between ECU nodes. And moreover, the ECUs nodes participating in communication need to be verified before taking part in communication. This chapter discusses the proposed certificate based ECU authentication mechanism followed by symmetric key generation, key distribution and key refreshment mechanisms.

4.1 Dependability

The dependability requirements as stipulated in ISO 26262 [8] requires that at least one critical fault must be tolerated by automotive application without loss of function. Contemporary automotive systems use single-core ECU that have difficulty in meeting the performance and dependability requirement. Technological advancement and low-cost of silicon-chips, we leverage multicore ECUs to provide FT. Our approach is equally applicable to dual-core and triple-core ECUs. Our multicore FT approach does not provide resilience against hardware failures, power supply failure, and other major component failures. However, these failures can be addressed by having redundant modules within the automotive.

We consider computing redundantly in multiple threads so that the results can be compared if any soft-error occur during computation. If we find any error during computation, recomputation is carried out until result match. The fault-tolerant approaches used are non-fault tolerant (NFT) or single threaded, FT-RMT, FT-RMT-QED, FT-RMT-CP. NFT implementation is a single threaded execution of the algorithm. FT-RMT implements two different threads in a multi-core process for the same program. FT-RMT-QED compares the execution results at multiple spots, detecting errors occurred during execution. FT-RMT-CP implements the lightweight checkpoint. In FT-RMT-CP, checkpoints are introduced in various portions of the code, dividing a program into small sections. When an error/fault occur in the program, the execution resumes from previous (last) checkpoint. This removes the need of re-executing whole program during errors.

4.2 Security Threat Model

The aim of this research project is to develop integrated dependability and security methodology for early design phase of a secure and dependable automotive. Assuming an adversary has gained access to intra-vehicular network and has plenty of time to access the network, the security model should prevent the passive and active attacks. Attackers can do passive eavesdropping which means they can sniff and steal all the traffic information from intra-vehicular network, losing the critical information about driver, vehicle and traffic will put the driver and passenger in great risk. Suppose that, in a x-by-wire systems, if an adversary knows the initial location of vehicle, and eavesdrops the communication of steering collecting the information of steering angle, acceleration, brake value etc, the adversary can easily know the final destination. Moreover, the attacker can do active eavesdropping and inject the false messages. If the adversary enters into intra-vehicular network, s/he can easily modify the content of message as well as inject fake messages into the network. This also can bring serious problems.

To address the threat of these active and passive attacks we can implement security primitives: encryption for confidentiality to discourage active attacks and message authentication codes for authentication to discourage passive attacks (message injection) [3], [4] and [17]. These approaches use symmetric key based cryptography because of less computation compared to public key cryptography. These approaches need to have symmetric key saved into secured memory. However, an adversary can attack on memory to get symmetric key. Losing symmetric key will compromise not only the single ECU or single vehicle but also compromises whole series of vehicle, because same series of ECU tend to use same symmetric key. Furthermore, if all the instructions are send using same symmetric key, an attacker can send same message resulting replay attacks. So, new symmetric key should be generated during driving to prevent attacks through memory and symmetric key should be updated to prevent replay attacks.

4.3 Cardinal Ingredients of the Proposed Approach

Figure 4.1 shows the overview of all entities that participate in authentication, symmetric key generation, and node-to-node communication.

As shown in Figure 4.1, CA is certificate authority that generates the necessary keys for all ECUs. CA has its own public key and private key. The public key is shared with all ECU nodes, whereas the private key is stored secretly. CA can be OEM, so the private key can be easily kept secret.

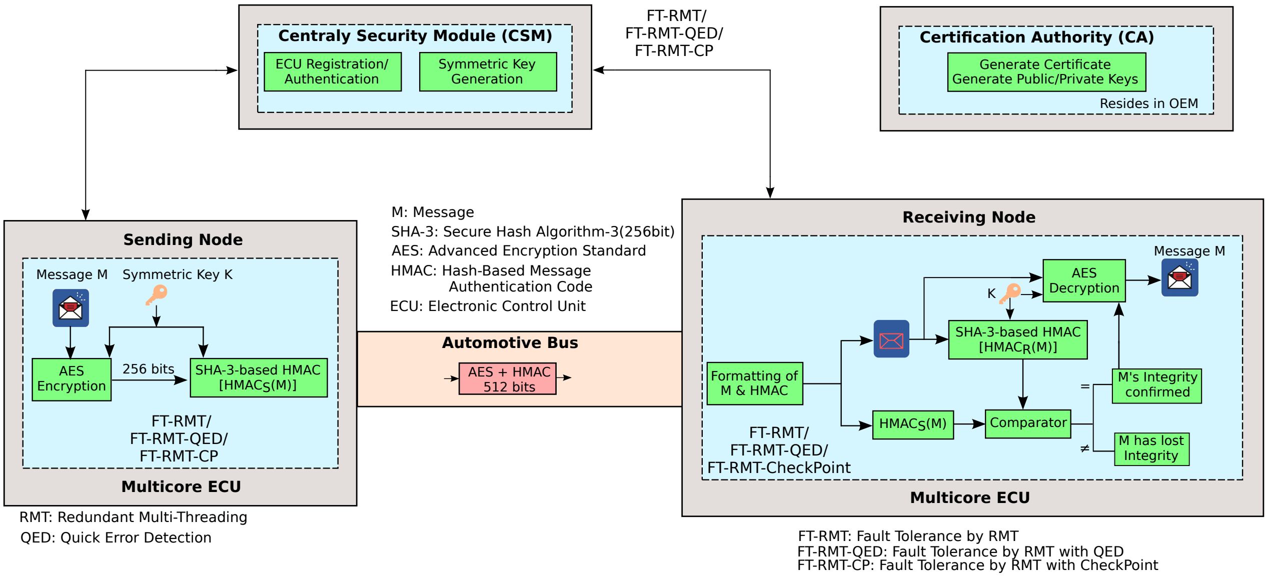
Centraly Security Module (CSM)
ECU Registration/ Authentication
Symmetric Key Generation
FT-RMT/
FT-RMT-QED/
FT-RMT-CP
Certification Authority (CA)
Generate Certificate
Generate Public/Private Keys
Resides in OEM
M: Message
SHA-3: Secure Hash Algorithm-3(256bit)
AES: Advanced Encryption Standard
HMAC: Hash-Based Message Authentication Code
ECU: Electronic Control Unit
Sending Node
Message M
Symmetric Key K
AES Encryption
256 bits
SHA-3-based HMAC [HMACS(M)]
FT-RMT/
FT-RMT-QED/
FT-RMT-CP
Multicore ECU
Automotive Bus
AES + HMAC 512 bits
Receiving Node
Message M
K
AES Decryption
SHA-3-based HMAC [HMACR(M)]
Formatting of M & HMAC
HMACS(M)
Comparator
=
M's Integrity confirmed
≠
M has lost Integrity
FT-RMT/
FT-RMT-QED/
FT-RMT-CheckPoint
Multicore ECU
RMT: Redundant Multi-Threading
QED: Quick Error Detection
FT-RMT: Fault Tolerance by RMT
FT-RMT-QED: Fault Tolerance by RMT with QED
FT-RMT-CP: Fault Tolerance by RMT with CheckPoint

Figure 4.1: *An Integrated dependable and secure approach (IDSA) for in-vehicle network*

Central Security Module (CSM) is responsible for ECU registration, authentication and symmetric key establishment. The CSM has its private key, public key, certificate signed by the CA, and a unique ID. It also contains public key of CA which is used to authenticate other ECUs. The sending node and receiving node are two sample ECU nodes of the vehicular network which participate in the communication. Each node has its private key, public key, certificate signed by the CA and a unique ID. Each ECU node also has the public key of CA which is used to authenticate the legitimate CSM. Although our proposed methodology requires secure storage of private key but security sensitivity of private key is much less than symmetric keys used for node-to-node communication. If a private key of an ECU is compromised, it only compromises one ECU, and upon revocation of the certificate of that ECU, no further harm can be done by the ECU. However, if symmetric key of an ECU is compromised, it has the potential to compromise all the ECUs in the vehicle or even the series of the vehicle because likely the same symmetric key would be programmed/stored by the OEM in all of the ECUs of the vehicle or even the series of vehicle.

The following subsections provide the overview of different stages of proposed work.

4.3.1 Certificate Generation

Suppose n_E be the total number of ECUs on the automotive bus. Each $ECU_i \ \forall \ i \in$ {1,2,3,...,n_E}, has a key denoted as

$$k_{E_i} = (k_{pub,E_i}, k_{pr,E_i}), \tag{4.1}$$

where k_{pub,E_i} is public key and k_{pr,E_i} is private key. The CA is responsible for generating these public and private keys for each ECU. The CA also generates the certificate for each ECU_i by combining ECUs public key k_{pub,E_i} with its identity ID_{E_i} and signature signed by the private key of the CA. The ID of each ECU is assigned by OEM (e.g., an ECU's serial number can serve as the ECU's ID).

The certificate ECU_i can be denoted as

$$Cert_{E_i} = [\,(\,k_{pub,E_i}, ID_{E_i})\,, S_{E_i}] \tag{4.2}$$

where, signature

$$S_{E_i} = sig_{k_{pr},CA}\,(k_{pub,E_i}, ID_{E_i}) \tag{4.3}$$

The algorithm for signature generation is given in Section 4.6.1.

4.3.2 ECU Authentication

Authentication of ECU can be done by certificate verification. Certificate has two parts: first, public key and ID of node k_{pub,E_i}, ID_{E_i}, and second, signature on the first part by using the private key of the CA $sig_{k_{pr},CA}\,(k_{pub,E_i}, ID_{E_i})$. By using $k_{pub,CA}$, we can verify if the signature is from a legitimate source or not. The algorithm for signature verification is given in Section 4.6.1.

4.3.3 Symmetric Key Generation and Establishment

After verifying all the ECUs, the CSM generates new symmetric key and distributes it to all ECUs. After certain time period T, this process of new key generation and establishment is repeated. The value of T is defined during manufacturing. The value of T should be greater than summation of time to authenticate all ECUs and the time to distribute symmetric keys to all ECUs. The time period T relates to desired symmetric key freshness interval. The process of key generation is described in Section 4.6.2.

4.4 Proposed Symmetric Key Establishment Protocol

Figure 4.2 presents our proposed protocol for symmetric key establishment. At the vehicle start up, the CSM advertises/broadcasts its public key, ID, and certificate to all other ECUs (step 1). This is done only once during start up.

All ECUs verify the authenticity of CSM. The information sent during this stage are not encrypted and an intruder can easily get the data but losing this information would not weaken the security of the network. After CSM authentication, the ECU i which requires registration or authentication generates a random nonce r_i.

For registration/authentication of ECU i, it's certificate and nonce should be sent to CSM. The message sent here should be encrypted because only the CSM should be able to retrieve the certificate and nonce of the ECU i and no other malicious ECU should be able to obtain this information. So, we generate a common secret between private key of ECU i and public key of CSM, which is transformed to obtain local key k_local. This local key can be generated only by ECU i and CSM, no other entity can generate this key. This key is used to encrypt the message and generate HMAC of the message $(m^r_{E_i}||h_m)$ (Step 2 ECU side). Here, HMAC is appended along with message to maintain integrity of the message.

When ECUs are authenticating the CSM, the CSM generates a new symmetric keys k_{sym} and the lifetime T for k_{sym} as prescribed by OEM (step 2 CSM side).

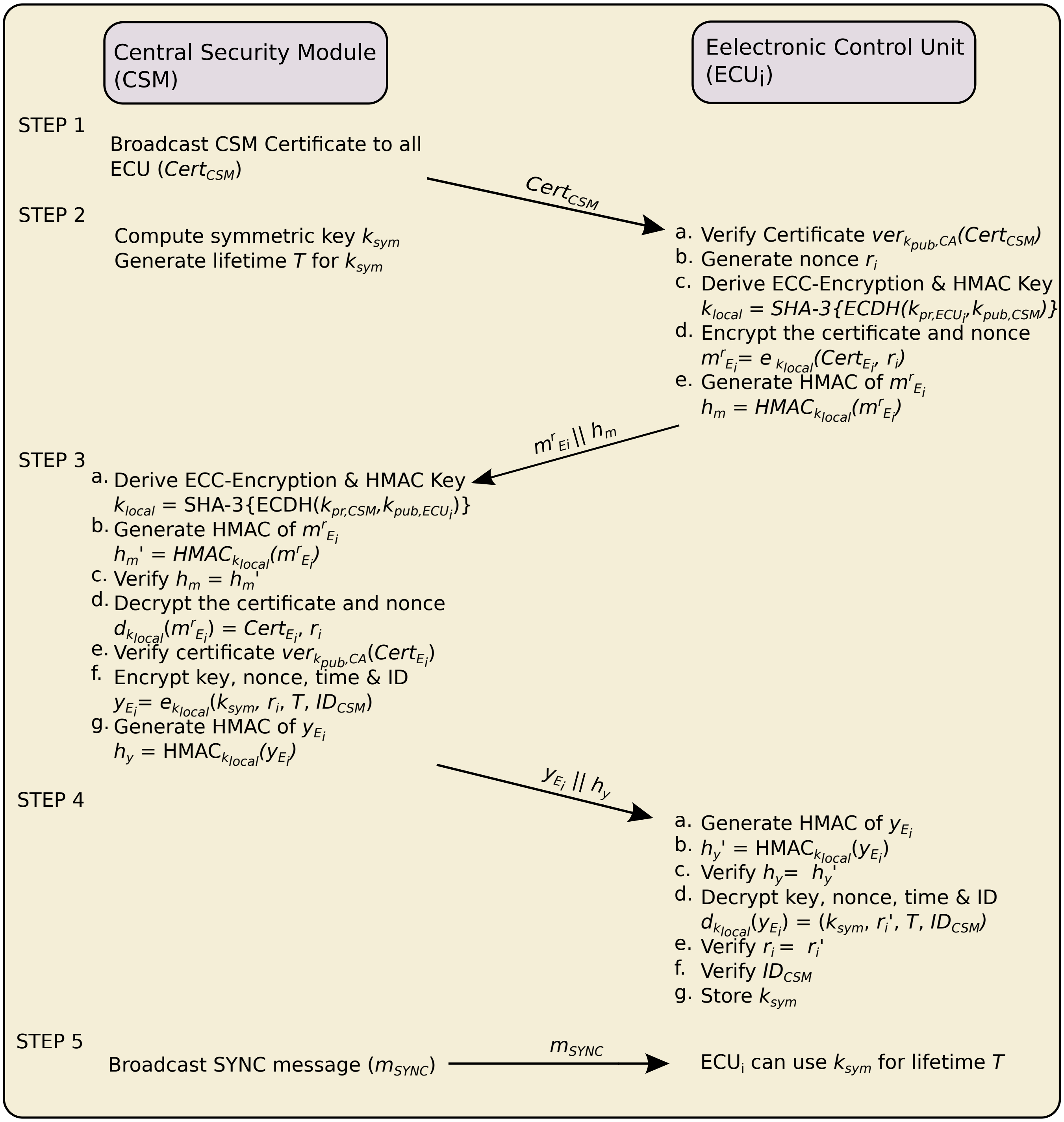
Central Security Module (CSM)
Eelectronic Control Unit (ECU_i)
STEP 1
Broadcast CSM Certificate to all ECU ($Cert_{CSM}$)
$Cert_{CSM}$
STEP 2
Compute symmetric key k_{sym}
Generate lifetime T for k_{sym}
a. Verify Certificate $ver_{k_{pub,CA}}(Cert_{CSM})$
b. Generate nonce r_i
c. Derive ECC-Encryption & HMAC Key
$k_{local} = SHA\text{-}3\{ECDH(k_{pr,ECU_i}, k_{pub,CSM})\}$
d. Encrypt the certificate and nonce
$m^r_{E_i} = e_{k_{local}}(Cert_{E_i}, r_i)$
e. Generate HMAC of $m^r_{E_i}$
$h_m = HMAC_{k_{local}}(m^r_{E_i})$
$m^r_{Ei} || h_m$
STEP 3
a. Derive ECC-Encryption & HMAC Key
k_{local} = SHA-3{ECDH($k_{pr,CSM}, k_{pub,ECU_i}$)}
b. Generate HMAC of $m^r_{E_i}$
$h_m' = HMAC_{k_{local}}(m^r_{E_i})$
c. Verify $h_m = h_m'$
d. Decrypt the certificate and nonce
$d_{k_{local}}(m^r_{E_i}) = Cert_{E_i}, r_i$
e. Verify certificate $ver_{k_{pub,CA}}(Cert_{E_i})$
f. Encrypt key, nonce, time & ID
$y_{E_i} = e_{k_{local}}(k_{sym}, r_i, T, ID_{CSM})$
g. Generate HMAC of y_{E_i}
h_y = HMAC$_{k_{local}}(y_{E_i})$
$y_{E_i} || h_y$
STEP 4
a. Generate HMAC of y_{E_i}
b. h_y' = HMAC$_{k_{local}}(y_{E_i})$
c. Verify $h_y = h_y'$
d. Decrypt key, nonce, time & ID
$d_{k_{local}}(y_{E_i}) = (k_{sym}, r_i', T, ID_{CSM})$
e. Verify $r_i = r_i'$
f. Verify ID_{CSM}
g. Store k_{sym}
STEP 5
Broadcast SYNC message (m_{SYNC})
m_{SYNC}
ECU_i can use k_{sym} for lifetime T

Figure 4.2: *Proposed symmetric key establishment protocol*

After receiving the request message ${m^r}_{E_i}$ along with hash h_m. The CSM verifies the HMAC to find out integrity of the message. Here, to calculate HMAC, we need to have local key k_local which was used in ECU side (step 2). The local key is again generated via common secret of private key of CSM and public key of ECU i. This key is used to generate HMAC, decryption and later for encryption. After verification of HMAC, the CSM decrypts the message and verifies the certificate $Cert_{E_i}$ using public key of CA. After the verification, the CSM encrypts the generated symmetric keys k_{sym}, nonce r_i, lifetime T and ID of the CSM ID_{CSM} with the public key of ECU i. This response message y_{E_i} is sent back to the ECU i (step 3) along with HMAC of message as h_y.

The received message packet y_{E_i} is tested for message integrity and then decrypted by the ECU i using its local key (generated in step 2). The ECU i verifies the random nonce r_i received from the CSM and its original random nonce. Further, the ECU i verifies ID_{CSM} with the one received in step 1. If verification is succeed then the ECU accepts the symmetric keys (step 4).

Finally, the CSM broadcast SYNC message instructing all ECUs to use the newly generated symmetric key for time T (step 5). Each ECU starts communicating with other nodes using this key for symmetric encryption and HMAC. After time period T, each ECU requests for a new key by sending a message request as in step 2 ECU side Figure 4.2. The CSM then distributes symmetric keys to ECUs as shown in Figure 4.2.

This protocol assures the timeliness through the following measures: the protocol specifies a lifetime T for the symmetric keys during which the symmetric keys are used for encryption communication; the ECU i sends the random nonce r_i to the CSM, and requires the CSM to encrypt r_i; if the returned value r'_i after decryption, matches the sent value r_i then ECU i will be assured that response came for that particular request. Even if an intruder knew the reply message packet to ECU i for now, it cannot be used to later perform replay attacks. Furthermore, including ID_{CSM} in response message from the CSM provides extra assurance to ECU i that it communicated with the authentic CSM.

The length of random nonce r_i must be long enough to resist the series of replay attacks. A malicious ECU can act as the CSM and can replay with old symmetric key for every request of new symmetric key. An ECU distinguishes by checking random nonce. However, if the length of random nonce is too small, random nonce may repeat after some runs and malicious ECU will be successful to deceive by sending old symmetric key . So, the length of random nonce should be long enough to survive series of replay attacks.

The synchronization mechanism is discussed in subsection 6.5.

We use elliptic curve cryptography (ECC) for the computation of steps 2, 3 and 4 explained above. The details of elliptic curve is given in Section 6.1. The basic information of elliptic curve is include in Section 3.3. The algorithm for certificate verification using signature verification is given in Section 4.6.1. The algorithm used in step 2(c, d and e), step 3(a, b, c, d) are steps of elliptic curve integrated encryption scheme (ECIES). The algorithm for ECIES is shown in Section 4.6.3.

4.5 Regular In-Vehicle Operation

ECU nodes communicate with each other through an authenticated encrypted message. The cipher text of the plain message is produced by using 128-bit block-based AES encryption. HMAC digest of the cipher text is computed and attached along with the cipher text. The sender ECU node sends the combination of cipher text and HMAC digest. The receiver ECU node extracts the cipher text and computes the MAC digest. The MAC digest computed at the receiver node is compared against the MAC digest sent along with the cipher text. If the MAC digest matches, the message is authentic otherwise the message has lost integrity and should be sent again. The details of regular operation is described in [17].

4.6 Algorithms used to Implement Proposed Key Exchange Protocol

In this section, we discuss different ECC-algorithms used to implement the protocol discussed in Figure 4.2.

4.6.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

This section describes the general idea of generating and verifying digital signature using elliptic curve. The following algorithms generate and verify signature which are used in certificate generation (Section 4.3.1) and ECU authentication using certificate verification(Section 4.3.2).

An elliptic curve ε with modulus p, coefficient a and b, and a generation point G is considered as shared ECC parameters. The CA has private key $k_{pr,CA}(= d)$ and public key $k_{pub,CA}(= K)$. The algorithm to generate signature by CA is shown in Algorithm 4. At first, CA generates a random number k_e less than the order of ECC $q(= \#E)$. Then, CA computes the scaler multiplication of k_e with generation point G resulting point as $R = (R_x, R_y)$. The x-component of $r(= R_x)$ is used to compute s as shown in step 4. Here $h(x)$ is hash of message x, (where, $x = (k_{pub,E_i}, ID_{E_i})$). The length of hashed message $h(x)$ must be at least as long as q. The final result produced by signature generation algorithm is concatenation of x-component of R 'r' and signature 's' which is denoted by '$r||s$'.

Algorithm 4: ECDSA Signature Generation Algorithm

```
Input: Elliptic curve $\varepsilon$,
prime number $p$,
coefficients of curve $a$ and $b$,
generation point $G = (G_x, G_y)$
order of curve $q = \#E$
message $x$
hash digest of message $h(x)$
private key of Signature Generator $k_{pr} = d \in (1, q)$
public key of Signature Generator $k_{pub} = d \cdot G = K$
Output: $r||s$, x-component of $R$ and signature $s$
1 Choose an integer as random ephemeral key $k_e \in (1, q)$
2 Compute $R = k_e \cdot G$
3 Let $r = R_x$
4 Compute $s \equiv (h(x) + d \cdot r)k_e^{-1}\ mod\ q$
```

The basic algorithm for signature verification is shown in Algorithm 5. ECU extracts signature s and x-component of random number r, and message x, (where, $x = (k_{pub,E_i}, ID_{E_i})$) from certificate. ECU also calculates $h(x)$ by using hash of x. w, u_1, u_2, T are calculated as

Algorithm 5: ECDSA Signature Verification Algorithm

```
Input: Elliptic curve $\varepsilon$,
prime number $p$,
coefficients of curve $a$ and $b$,
generation point $G = (G_x, G_y)$
order of curve $q = \#E$
message $x$
hash digest of message $h(x)$
x-component of random number $r$
signature $s$
public key of Signature Generator $k_{pub} = K$
Output: Valid or Invalid Signature
Compute value $w \equiv s^{-1}\ mod\ q$
Compute value $u_1 \equiv w \cdot h(x)\ mod\ q$
Compute value $u_2 \equiv w \cdot r\ mod\ q$
Compute value $T = u_1 G + u_2 K$
The verification rule is:
    $x_T \begin{cases} \equiv r\ mod\ q \implies valid\ signature \\ \not\equiv r\ mod\ q \implies invalid\ signature \end{cases}$
  where, $x_T$ is x-component of $T$
```

shown in step $1 - 4$. The x-component of T i.e. x_T is compared with r, if they are equal it is a valid signature otherwise it is a invalid signature.

4.6.2 Symmetric Key Generation Process

For symmetric key generation, the CSM selects any random number $R_{num} \in \{2, 3...., \#E\}$, where $\#E$ denotes the total number of points in elliptic curve. R_{num} is multiplied with generation point $G = (G_x, G_y)$ to get some other point (k_x, k_y) which is then hashed by SHA-3 256-bit algorithm. Here, k_x is only hashed to get 256-bit hashed value. The first 128-bit is used for symmetric key encryption and decryption, as well as for message authentication HMAC. The generated keys are then distributed to all other ECUs for communication. After certain period T this process is repeated and new keys are generated and distributed.

4.6.3 Elliptic Curve Integrated Encryption Scheme (ECIES)

In step 2 and step 3 as shown in Figure 4.2, the message packets are encrypted and send. Here, the encryption scheme used is ECIES which is explained in this section.

ECIES is the hybrid encryption and decryption scheme based on ECC. We use different functions for ECIES which are:

- Key Agreement Function(KAF): Function used to generate shared keys between two parties. We use ECDHA as KAF.

- Key Derivation Function (KDF): Mechanism used to extract the key form set of keys. We use hash of x-component of keys produced by KAF.

- Encryption and Decryption (ENC & DEC): Symmetric encryption algorithm. AES-128 bit encryption and decryption scheme is used.

- Message Authentication Code (MAC): Data used to authenticate the message. Hash based MAC is generated using hash function.

- Hash (HASH): Digest function, used within KDF and the MAC functions. SHA-256 *kecckak-p* hash algorithm is used.

Let us follow the tradition, Alice wants to send a message to Bob. In public key cryptography approach Alice and Bob have their private and public keys say, Alice has private key say 'a', public key 'A' and Bob has private key 'b' and public key 'B'. Both Alice and Bob agree on using an elliptic curve ε, with constants a and b and generation point $G = (G_x, G_y)$.

The basic idea of ECIES algorithm is to use elliptic curve deffie-hellman key exchange (ECDH) to exchange common secret. Apply some key derivation function (like hash functions) on common secret to generate symmetric key. This symmetric key is then used in symmetric encryption, decryption and producing HMAC.

The Algorithm 6 and 7 are given below for encryption and decryption using ECIES scheme respectively.

Algorithm 6: ECIES Encryption Algorithm

```
Input: Elliptic curve $\varepsilon$,
plain message $m_p$
private key of Alice $a$
public key of Bob $B$
Output: cipher message with its MAC $m_c||m_{c,MAC}$
generate shared secret using KAF $k_{sec,x}$ where $k_{sec} = (k_{sec,x}, k_{sec,y}) = a \cdot B$
use a KDF to generate symmetric key for encryption and message authentication
  codes $k_{enc,mac} = HASH(k_{sec,x})$
encrypt the plain message $m_c = ENC(k_{enc,mac}, m_p)$
generate the MAC of cipher message $m_{c,MAC} = MAC(m_c)$
concatenate $(m_c||m_{c,MAC})$
```

Algorithm 7: ECIES Decryption Algorithm

```
Input: Elliptic curve $\varepsilon$,
cipher message with MAC $m_c||m_{c,MAC}$
private key of Bob $b$
public key of Alice $A$
Output: plain message $m_p$
generate shared secret using KAF $k_{sec,x}$ where $k_{sec} = (k_{sec,x}, k_{sec,y}) = b \cdot A$
use a KDF to generate symmetric key for encryption and message authentication codes $k_{enc,mac} = HASH(k_{sec,x})$
generate the MAC of cipher message $m^{'}_{c,MAC} = MAC(m_c)$
check MAC tag with generated MAC, decryption fails if $m^{'}_{c,MAC} \neq m_{c,MAC}$
decrypt the plain message $m_p = DEC(k_{enc,mac}, m_c)$
```

Chapter 5

Case Study: Steer-by-wire Subsystem

In a SBW subsystem, the heavy mechanical steering column is substituted by electronic system. It has two advantages-first, SBW subsystem eliminates the risk of steering column entering the cockpit during frontal crash. Second, steering column is one of the heaviest part of vehicle, its removal reduces the weight of vehicle. Less weight lowers the fuel consumption. In this chapter, we explain our SBW case study that leverages dual-core ECUs to incorporate dependability and security primitives.

5.1 Steer-by-wire Operational Architecture

The SBW subsystem provides the same functionalities as conventional steering column: front axle control (FAC) and hand-wheel (HW) force feedback (HWF). The front axle control controls the wheel direction according to hand-wheel whereas, hand-wheel force feedback provides the mechanical like feedback to hand-wheel. The basic architecture of SBW is shown in Fig 5.1. The rotation on hand-wheel is sensed by hand-wheel sensors and the sensed values are fed as the input to HWS ECU1. HWS ECU1 processes the information and creates CAN packet with processed hand-wheel sensor information. The CAN packet is then transmitted to FAA ECU1 via CAN bus. The FAA ECU1 processes CAN packet and turns the actuators to rotate the wheels based on the command information in th received

CAN packet. The wheel rotation effect is sensed by front axle sensors and processed by FAS ECU2 and a CAN packet is transmitted to HWA ECU2 as response. The HWA ECU2 turns actuators to provide force feedback to hand-wheel. The connection between the ECU is by CAN bus. The sensors/actuators and the ECUs are connected by point-to-point links. Furthermore, the SBW subsystem is made FT by using dual-core ECUs, redundant-sensors and redundant-actuators. In this work, we only focus on front axle control (FAC) part to compute the response time and error resilience of our FT approach.

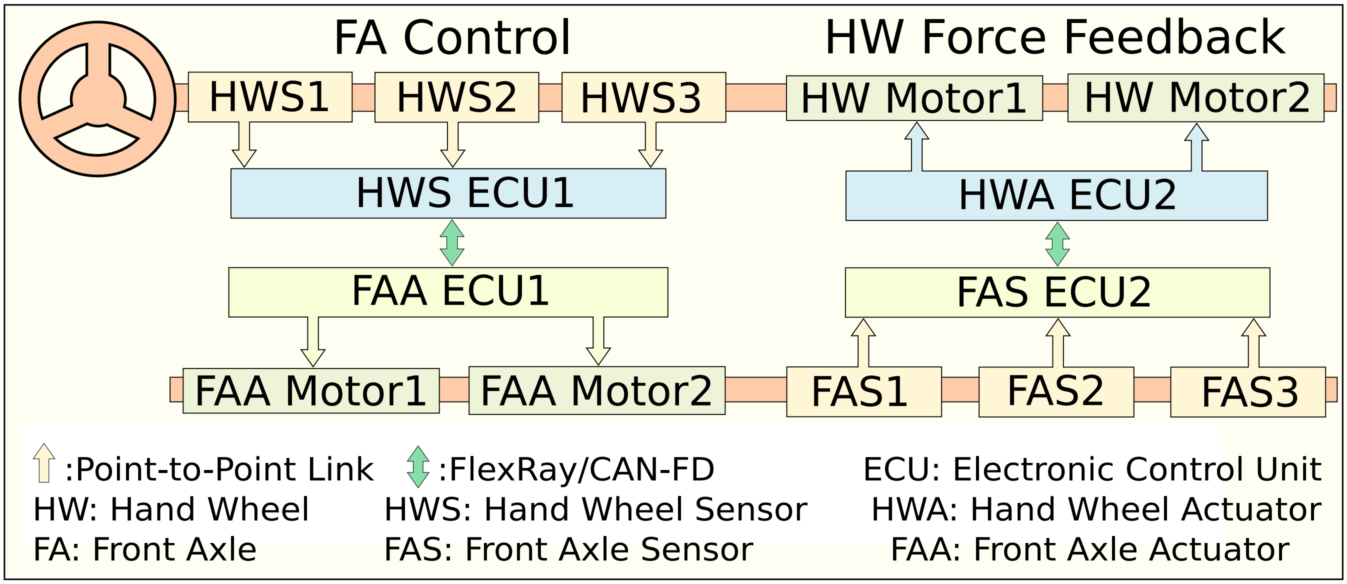

Figure 5.1: *Steer-by-wire Operational Architecture*

5.2 Timing Model of SBW

The delay between the drivers request at the HWS and the corresponding response at the front axle actuator (FAA) has significant impact on the working of SBW subsystem. The end-to-end delay/response (τ_r) is regarded as Quality of Service (QoS) metric. If this delay exceeds a critical threshold value ${\tau_r}^{max}$, it effects the safety and reliability of the automotive. This critical threshold value is defined by automotive OEMs. The probability that the worst-case response time is less than the critical threshold is termed as behavioral reliability. In the following, we analytically model the response time for the SBW subsystem and error resilience of our FT approaches. The response delayτ_r time is given by following equation,

$$\tau_r = \tau_p + \tau_m + \tau_s \tag{5.1}$$

where, τ_p is pure delay, τ_m is mechatronic delay, and τ_s sensing delay.

The mechatronic delay is introduced by the actuators (electric motor in our case). The sensing delay is the delay caused by sensors and the interaction of application processor of ECUs with the sensors. The sensing and mechatronic delays are bounded by a constant value of $3.5ms$ [18]. For our secure and dependable approach, the pure delay (τ_p) includes ECUs computational delay for processing the control algorithm, computational delay for processing the incorporated security and dependability primitives(depends on the execution time of the ECU), and transmission delay including bus arbitration (depends on the type of in-vehicle network used like CAN-FD or FlexRay). Mathematically, pure delay (τ_p) for our FAC function can be written as,

$$\tau_p = rcc1 \cdot \tau_{hws}^{ecu1} + rtc \cdot \tau_{bus} + rcc2 \cdot \tau_{faa}^{ecu1} \leq \tau_p^{max} \tag{5.2}$$

where τ_{hws}^{ecu1} and τ_{faa}^{ecu1} denote the computation time at HWS-ECU1 and FAA-ECU1, respectively; τ_{bus} represents the transmission time for a message on an in-vehicle bus (CAN FD, or FlexRay) from HWS-ECU1 to FAA-ECU1; $rcc1$ and $rcc2$ represent the number of re-computations that are needed to be done at HWS-ECU1 and FAA-ECU1, respectively, for error correction; rtc represents the number of retransmissions required for an error-free transmission of a secure message over in-vehicle bus; and τ_p^{max} represents maximum allowable τ_p. According to Wilwert et al. [19], with a minimum tolerable QoS score of 11.15, the critical limit for pure delay τ_p^{max} is $15ms$, beyond which the vehicle becomes unstable and risks the drivers safety.

Here, pure delay at FAC can be divided in two parts.

- **Delay during regular operation:** During regular operation the delay is only due to computation of encryption and HMAC for secured communication at both sending and receiving ends and delay due to message transmission.

$$\begin{aligned} \tau_p = rcc1 \cdot \tau_{hws}^{ecu1,RegOp} + rtc \cdot \tau_{bus} \\ + rcc2 \cdot \tau_{faa}^{ecu1,RegOp} \leq \tau_p^{max} \end{aligned} \tag{5.3}$$

where, $\tau_{hws}^{ecu1,RegOp}$, $\tau_{faa}^{ecu1,RegOp}$ denotes the delay at HWS-ECU1 and FAA-ECU1 respectively during regular operation.

- **Delay during key refreshment operation:** During key refreshment operation the delay is summation of delay during regular operation and delay due to key refreshment operation (one of the protocol step, since only one step is done during critical time period).

  So above equation can be rewritten as,

$$\begin{aligned} \tau_p = rcc1 \cdot \tau_{hws}^{ecu1,RegOp} + rcc3 \cdot \tau_{hws}^{ecu1,KeyRf} + rtc \cdot \tau_{bus} \\ + rcc2 \cdot \tau_{faa}^{ecu1,RegOp} + rcc4 \cdot \tau_{faa}^{ecu1,KeyRf} \leq \tau_p^{max} \end{aligned} \tag{5.4}$$

  where, $\tau_{hws}^{ecu1,KeyRf}$ and $\tau_{faa}^{ecu1,KeyRf}$ denotes delay at HWS-ECU1 and FAA-ECU1 respectively during key refreshment operation, $rcc3$ and $rcc4$ represent re-computation due to fault at HWS-ECU1 abd FAA-ECU1 respectively.

  Since, during key refreshment operation CSM handles only one ECU at a time, either HWS-ECU1 or FAA-ECU1 computation will be carried out at one time. So the Equation 5.4 will be

$$\begin{aligned} \tau_p = rcc1 \cdot \tau_{hws}^{ecu1,RegOp} + rcc \cdot \tau_{fac}^{ecu,KeyRf} + rtc \cdot \tau_{bus} \\ + rcc2 \cdot \tau_{faa}^{ecu1,RegOp} \leq \tau_p^{max} \end{aligned} \tag{5.5}$$

  where, $\tau_{fac}^{ecu1,RegRf}$ denotes key refreshment computation either at HWS or at FAA, rcc is total re-computation at either end during faults.

Chapter 6

Result and Discussion

In this chapter we present our experimental set up and evaluation results timing analysis.

6.1 Security Standards

ECC: For ECC implementation, we have used a prime field curve P-192 from NIST [20]. For the prime p, the pseudo-random curve is given as

$$\varepsilon : y^2 \equiv x^3 - 3ax + b(mod\, p) \tag{6.1}$$

of order n. The details of curve are given in Table 6.1.

AES: For implementing AES, we follow NIST FIPS-197 [21]. We implement AES-128 bit encryption and decryption.

Table 6.1: *NIST P-192 curve details*

Item		Value
prime modulus	(p)	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF
order	$(\#E)$	0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
a coefficient	(a)	-0x3
b coefficient	(b)	0x64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
x-generation point	(G_x)	0x188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012
y-generation point	(G_y)	0x07192B95FFC8DA78631011ED6B24CDD573F977A11E794811
cofactor	(h)	1

SHA3: For hashing, we use SHA3-256 bit based on FIPS-202 [22]. We follow $keccak - p$ algorithm.

HMAC: For message authentication codes we follow NIST FIPS-198-1[23]. For hashing we use SHA3-256 bit hash function. The key length used for HMAC is 128 bit.

6.2 Experimental Setup

We implemented the computational aspects of the proposed IDSA on *Cortex-A57* in NVIDIA's Jetson TX2 kit [24]. ARM Cortex-A57 has 64-bit quad core ARMv8 processor running *Ubuntu 14.04.4 LTS* at $2.0GHz$. All the codes for authentication, key-generation and key-distribution are written in C-language. OpenMp is used for FT-RMT implementation.

Operational Parameters: For our SBW subsystem, we assume the steering wheel sensor sampling rate to be fixed at $420Hz$, which corresponds to the sampling delay, τ_{sample} of, $2.38ms$ [18].

6.3 Timing Analysis

In this sub-section, we describe the timing response of key-generation and key-establishment process in different fault tolerant approaches. For timing analysis, we introduce soft errors at different points in the program. Our approach emulates bit flipping in the program/memory due to noise and/or radiation hitting the computing and/or memory elements.

6.3.1 Performance Overhead Due to Fault Tolerance Approaches

In this section, we present execution time taken by different steps of protocol (Figure 4.2) without error. We calculate the overhead caused by FT approaches.

Table 6.2 shows execution time of different steps of the key establishment protocol (Figure 4.2) in various operational modes. Table 6.2 does not contain computation time for step 1 and step 5 because in these steps the CSM broadcasts the precomputed message packet,

Table 6.2: *Timing of Key Generation and Establishment Steps in Different Operational Modes*

Operational Modes	Algorithm Steps (time in μsec)			
	Step 2		Step 3	Step 4
	ECU Side	CSM Side	CSM Side	ECU Side
NFT	5214.63	216.51	5224.71	1563.63
FT-RMT	5259.53	219.80	5226.57	1565.86
FT-RMT-QED	5306.83	220.42	5300.08	1571.29
FT-RMT-CP	5352.41	219.71	5300.93	1585.75

Table 6.3: *Effect of error location on performance (for step 2 ECU side)*

Error Location	Operational Modes (time in μsec)		
	NFT	FT-RMT-QED	FT-RMT-CP
@ Verification of Certificate	10223.79	8483.39	8760.96
@ ECC-Encryption (Gen Key)	10191.91	9951.42	6837.83
@ ECC-Encryption (AES-Encryption)	10202.22	10153.84	5692.02
@ ECC-Encryption (HMAC)	10323.72	10224.69	5747.74

which do not require computation. This table shows the computation time without any error in the program. Without the fault injection (i.e. in the absence of errors), execution time increases sequentially from NFT to FT-RMT to FT-RMT-QED to FT-RMT-CP. This increment is due to multiple reasons such as overhead of thread creation, variable comparison to detect error and overhead of storing temporary variables for check-pointing. In the table, we can see that the total overhead is very insignificant (less than 1%). This is because the computation time of the steps in the key establishment protocol are very large compared to the summation of all overheads.

6.3.2 Effect of Error Location on Performance

Table 6.3 shows the execution time of Step 2 ECU side. In this examination, single soft error is injected at different points in the program and observed. The fault-injection emulates bit flipping, in the program/memory due to external noise and/or radiation. If a single error occurs in FT-RMT mode, the entire function is recomputed to rectify the error. This results in the computation time to be 2× the computation time without the error. In FT-RMT-

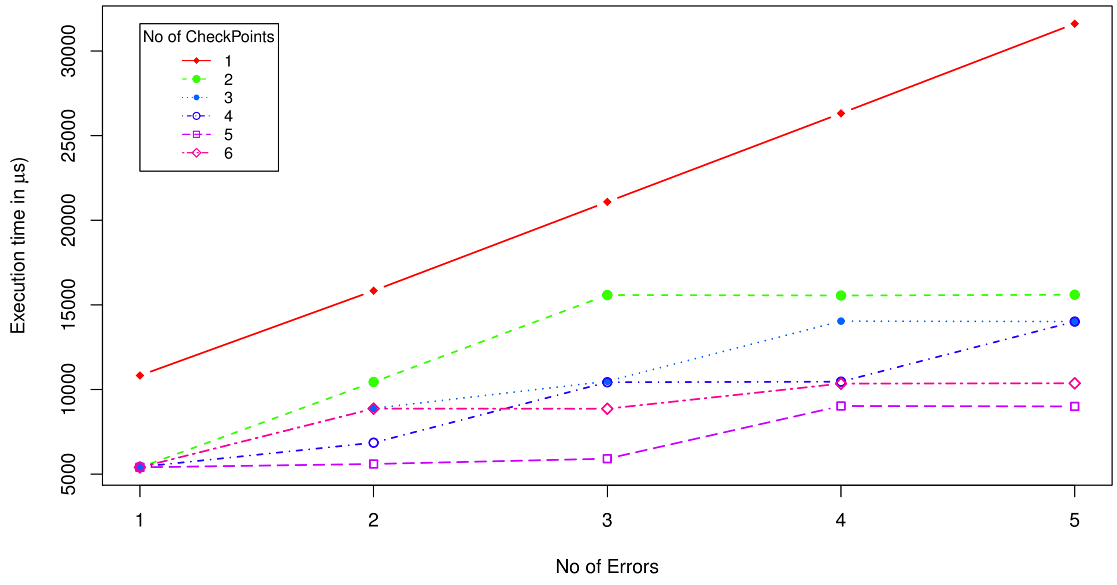

Figure 6.1: *Effect of checkpoint and error on performance (for step 3)*

QED, if error occurs during the start of program, the computation overhead is less compared to FT-RMT. However, if error occurs around the end of program, the computation overhead is almost the same as that of FT-RMT. FT-RMT-CP repeats the execution of those parts where error occurred. The recomputation overhead depends on the size of code between two consecutive checkpoints where error(s) has occured. Table 6.3 presents that overhead due to errors at Verification of Certificate step is comparatively more than the overhead due to errors at other locations of the program. This is because Verification of Certificate demands majority of execution time. From the table, we observe that the performance during faults is best in FT-RMT-CP.

6.3.3 Effect of Checkpointing on Performance with Errors

Figure 6.1 shows the execution time of the step 3 pf the key establishment protocol when multiple errors are injected at different points with varying number of checkpoints in the program. The number of checkpoints span from one to six, whereas number of errors introduced vary from one to five. The errors are evenly spread at different checkpoints of the program

Table 6.4: *Performace of Regular OPeration at Sender and Receiver ECU Nodes*

Operational Mode	Sender Node (time in μsec)	Receiver Node (time in μsec)
NFT	130	87
FT-RMT	151	109

in order for fair and comprehensive evaluation. Here, we consider that in one checkpoint only one error occurs. Two errors occur in one checkpoint means the first error occurs in first computation and second error occurs in second computation. Two errors occur in two checkpoints means two error occur in two different checkpoints.

In FT-RMT-CP, having a single checkpoint is same as FT-RMT model, where results are compared only at the end of program. In single checkpoint, when any error occurs whole function needs to be recomputed and as the number of errors increase the computational time increases linearly. If we increase the number of checkpoints, the time consumption to rectify the error decreases. From the Figure 6.1, it can be observed that the recomputation time also depends on the error location in the program. In 6-checkpoint evaluation, a drastic increase of computation time is observed when number of errors changed from 1 to 2. This is because the new error (from 1 to 2) was introduced in the compute-intensive region of the program. From this table, it can be observed that a large number of checkpoints in a program decreases the computation overhead in the presence of large number of soft-/transient errors. This is only applicable for programs with large computation time than the overhead of thread creation, -variable comparison, and temporary variable storage.

6.3.4 Performance Analysis of Regular Operation

Table 6.4 shows the temporal performance obtained from our experiments. The results show that the encryption and HMAC at the sender node takes longer time as compared to the decryption and HMAC at receiver node. This is because in receiver node the decryption computation is accelerated by using pre-computed tables. We observe FT-RMT at sender node has 16.2% overhead and receiver node has 25.2% overhead as compared to NFT.

Table 6.5: *ECU Nodes*

rcc1, rcc2, rtc	rcc			
	Step 2	Step 2	Step 3	Step 4
	ECU Side	CSM Side	CSM Side	ECU Side
rcc1 = 2, rcc2 = 2, rtc = 2	2.38	58.08	2.40	8.50
rcc1 = 3, rcc2 = 3, rtc = 3	2.17	52.87	2.18	7.32
rcc1 = 4, rcc2 = 4, rtc = 4	1.96	47.67	1.96	6.60

6.4 Feasibility Analysis

As described in section 5.2 the pure delay or real-time deadline for transmitting a message packet from sender node to receiver node in SBW subsystem is 15ms. This critical limit for real-time (15ms) constitutes of combined computation time from sender node to receiver node, as well as the message transmission time. As shown in Table 6.4, the computational time at sender node for encryption (2-blocks of 128-bit) and HMAC (256-bit digest) is 0.151ms, and the computational time at receiving node for decryption and HMAC is 0.109ms. The message transmission time using CAN-FD is 0.12ms (for a packet of 512 bits) [4]. The total computational and transmission delay without error is 0.38ms (for one 512-packet of CAN-message). Furthermore, in period of 15ms at least 6 readings are taken by sensor (Section 6.2). Considering one block-of-AES can store the reading of one sample, at least 3 CAN frame (1 frame holds 2 reading, and in 15ms there will be 6 readings) are transmitted within a period of 15ms without losing any sample value.

During faults in computation and transmission process, the program recomputes and retransmits the faulty computations and packets, respectively, at the corresponding sender, receiver nodes and/or transmission links. For this study, the different number of recomputation for each step of the protocol is shown in Table 6.5. For 1 recomutations (rcc1 = 2, rcc2 = 2) at receiver end and sender end as well as 1 retransmission (rtc = 2), we have enough time to recompute any step of the protocol. Similarly, number of recomputation at sender end and receiver end affecting recomputation of step of the protocol is shown in table.

Here, our key establishment protocol runs only one step during one critical time-period and CSM can serve to only one ECU at a time.

This data verifies the feasibility of our approach. Our approach meets the real-time deadline and feasible to use.

6.5 Synchronization

Synchronization of symmetric key at all ECUs is fundamental for proper functioning of the protocol. The key is used to encrypt the messages at the sender node must be available and used to decrypt the messages at the receiver node. Therefore, a synchronization mechanism needs to be established. CSM node stores the list of all ECUs nodes ID number. When the vehicle starts, the CSM authenticates and registers all ECU nodes. The CSM produces a new symmetric key along with the expiration period T and distributes it to all ECUs. As explained in the key establishment protocol (Figure 4.2), the CSM broadcasts SYNC message and directs all the ECUs to use new symmetric key. After receiving the SYNC message, all the ECU nodes use the newly established symmetric key for encryption, decryption, and HMAC operation.

The time period field T indicates the expiration time of the symmetric key. After time T, a new symmetric key is generated and established by the CSM. During the new key establishment process, all the ECUs function with the current symmetric key. The CSM has the knowledge of total number of ECU nodes, and broadcasts the SYNC message to all the ECUs (Step 5). As soon as all the ECUs receive the SYNC message, the ECUs copy the new key to the current key field. If an ECU node is decrypting the message it continues without halting. However, if an ECU node is encrypting the message, it stop its execution and resumes execution with new symmetric key. Here, the receiver ECU does not stop the decryption process as the previously transmitted message by the sender ECU must be decrypted using the old symmetric key. The new key is used for an on-going encryption because the receiver node will update its symmetric key.

Chapter 7

Conclusion

We propose a novel ECUs authentication and symmetric key establishment protocol for intra-vehicle network of automotive CPS. We realize ECU authentication using certificate-based authentication. Key-establishment mechanism establishes a common symmetric (session) key for all ECUs intra-vehicular network to implement secured communication over the network. This protocol removes the need of storing symmetric keys in ECUs memory permanently. Moreover, the study incorporates key refreshment policy for each symmetric (session) key, which are updated after certain lifetime/timeframe. We leverage multi-core ECUs to embed fault-tolerance by using redundant multi-threading (FT-RMT), performs quick error detection (FT-QED) and accelerate performance using lightweight checkpointing (CP).

We have implemented our proposed protocol in *Cortex-A57* on NVIDIA TX2 kit. We demonstrate the effectiveness of our proposed approach using a SBW application over CAN FD as a case study. We show that our method provides ECU authentication, symmetric key generation and distribution in intra-vehicle ECU communication without violating safety and real-time constraints of the vehicle. Moreover, results reveal that our fault tolerant approach has very minimal overhead ($< 1\%$) and among FT-RMT, FT-RMT-QED and FT-RMT-CP, the lowest overhead of recomputation during fault occurrence is of FT-RMT-CP.

Bibliography

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, *et al.*, “Experimental security analysis of a modern automobile,” in *Security and Privacy (SP), 2010 IEEE Symposium on*, pp. 447–462, IEEE, 2010.

[2] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, p. 91, 2015.

[3] A. Munir and F. Koushanfar, “Design and performance analysis of secure and dependable cybercars: A steer-by-wire case study,” in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pp. 1066–1073, IEEE, 2016.

[4] B. Poudel, N. K. Giri, and A. Munir, “Design and comparative evaluation of gpgpu-and fpga-based mpsoc ecu architectures for secure, dependable, and real-time automotive cps,” in *Application-specific Systems, Architectures and Processors (ASAP), 2017 IEEE 28th International Conference on*, pp. 29–36, IEEE, 2017.

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

[6] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Design Automation Conference, 2007. DAC’07. 44th ACM/IEEE*, pp. 9–14, IEEE, 2007.

[7] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners.* Springer Science & Business Media, 2009.

[8] I. O. for Standardization, “ISO 26262 road vehicles - functional safety,” Feb 2016.

[9] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra, "Qed: Quick error detection tests for effective post-silicon validation," in *Test Conference (ITC), 2010 IEEE International*, pp. 1–10, IEEE, 2010.

[10] R. M. Ishtiaq Roufa, H. Mustafaa, S. O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, and I. Seskarb, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study," in *19th USENIX Security Symposium, Washington DC*, pp. 11–13, 2010.

[11] T. Huang, J. Zhou, Y. Wang, and A. Cheng, "On the security of in-vehicle hybrid network: Status and challenges," in *International Conference on Information Security Practice and Experience*, pp. 621–637, Springer, 2017.

[12] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-security for the controller area network (can) communication protocol," in *Cyber Security (CyberSecurity), 2012 International Conference on*, pp. 1–7, IEEE, 2012.

[13] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *International Conference on Information Security and Cryptology*, pp. 302–318, Springer, 2011.

[14] E. Beckschulze, F. Salewski, T. Siegbert, and S. Kowalewski, "Fault handling approaches on dual-core microcontrollers in safety-critical automotive applications," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pp. 82–92, Springer, 2008.

[15] U. Keskin, "In-vehicle communication networks: a literature survey," *Computer Science Report*, vol. 10, 2009.

[16] C. Specification, "Version 2.0," *Robert Bosch GmbH*, 1991.

[17] A. Munir and F. Koushanfar, "Design and analysis of secure and dependable automotive cps: A steer-by-wire case study," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[18] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, “Timing modeling and analysis for autosar-based software development: a case study,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 642–645, European Design and Automation Association, 2010.

[19] C. Wilwert, Y.-Q. Song, F. Simonot-Lion, and T. Clément, “Evaluating quality of service and behavioral reliability of steer-by-wire systems,” in *9th IEEE International Conference on Emerging Technologies and Factory Automation-EFTA'2003*, vol. 1, pp. 193–200, IEEE, 2003.

[20] P. FIPS, “186-4,” *Digital Signature Standard (DSS)*, 2013.

[21] N. F. Pub, “197: Advanced encryption standard (aes),” *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[22] M. J. Dworkin, “Sha-3 standard: Permutation-based hash and extendable-output functions,” tech. rep., 2015.

[23] P. FIPS, “198-1,” *The Keyed-Hash Message Authentication Code (HMAC)*, 2008.

[24] N. Corporation, “Data sheet nvidia jetson tx2 systme-on-module,” 2017.