Secure Intra-Vehicular Communication over CANFD

Ali Shuja Siddiqui[1], Chia-Che Lee[3], Wenjie Che[2], Jim Plusquellic[2] and Fareena Saqib[1]
[1]*Dept. of Electrical and Computer Engineering, University of North Carolina Charlotte, North Carolina, U.S.A.*
[2]*Dept. of Electrical and Computer Engineering, University of New Mexico, New Mexico, U.S.A.*
[3]*Mindtree Limited, Redmond, Washington, U.S.A.*
[1]{asiddiq6, fsaqib}@uncc.edu, [2]{wjche@, jimp@ece.}unm.edu, [3]{chiache.lee@mindtree.com}

Abstract— Recent documented remote hijacking attacks on vehicles have created a need for improving on-board vehicular network security. Electronic Control Units (ECUs) in a vehicle are resource constrained devices that are connected using network standard such as Controller Area Network (CAN). In our work, we demonstrate threat models of CANBus, and propose a secure and trusted framework that implements lightweight hardware based authentication and secure key exchange for enhanced security over the insecure CANFD standard. The scheme integrates a hardware based code isolation using Trusted Execution Environment that further restricts the access to crypto IPs and resources at the time of device or network compromise. The paper discusses the overhead, performance and security analysis of our proposed framework.

Keywords—Hardware Security, Physical Unclonable Functions PUF, Elliptic Curve Cryptography ECC, CANFD

I. Introduction

Advancements in vehicle electronics including sensors and state of art control units have contributed in the progress of automotive industry. In the past, the role of electronics was limited to supporting the motor functions in vehicles, but as computing systems have evolved, electronic subsystems have become an integral part of an automobile. One common application is in the form of Electronic Control Units (ECUs). ECUs are electronic controllers attached to mechanical subsystems in a car and are used for actuation and reading telemetry information from the subsystem it is connected to. To exchange data, ECUs are connected in a computer network like fashion. There are various network standards that are used in the automotive industry, such as Controller Area Network (CAN), that works on differential signaling, FlexRay Bus, that can communicate at speed of 10Mbps and local interconnect network protocol (LIN) for low cost devices. Controller Area Network (CAN) is the one of the most widely used standard. We demonstrate threat models of CAN Bus protocol in detail and present a secure communication framework over CAN Bus with enriched electronic control unit ECU design. Section 2 provides a background study of CAN Bus, threat model, trusted execution environments and physical unclonable function PUF. In Section 3, framework design and implementation are discussed. Section 4 details performance analysis, and in Section 5 security analysis is presented.

This research has been sponsored by the National Science Foundation under grant no. 1566530 and 1623299.

978-1-5386-1421-1/17/$31.00 ©2017 IEEE

II. Background Studies

A. Controller Area Network

The Controller Area Network was developed by Bosch GmbH and later became ISO Standard 11898 in 1993. Over the years, there have been additions to the standard, with the latest addition CAN Flexible Data-rate CANFD. CANFD was introduced in the year 2015 as a review of the standard ISO 11898-1[1].

CAN is a multi-master bus network, where multiple nodes can initiate communication. CAN Bus connects devices using differential pair of wires CANH and CANL. CAN Bus is a low cost means to establish a resilient communication network. The traffic on bus is visible to all connected nodes. CAN Bus implementation supports three connection speeds 125 kbps, 500 kbps and 1 Mbps. In a CAN, all nodes are set to transmit and receive at the same data rate. If any node tries to transmit data on a different speed, the data on the entire bus is corrupted. However, in the CANFD standard, a node can transmit data at a maximum of 12 Mbps. The data and the following CRC (checksum) bits of a frame can be sent at a pre-decided shifted data rate, whereas the rest of the frame is still transmitted on standard CAN speeds.

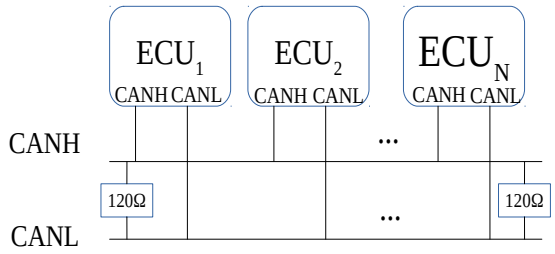

Figure 1: CAN Bus Physical Connection.

In the classical CAN Bus standard, there are two lengths of CAN data frames, *Standard* and *Extended*. A standard data frame is defined in figure 2. The Arbitration field contains 11 bits for the target identifier and one bit represents if it is a request frame. A frame does not include source address. In the Extended CAN bus frame, the length of the identifier is of 29 bits. The Data field can hold up to 64 bits of data. The length of the standard CAN frame is 108 bits whereas 131 for an extended CAN frame. On the physical medium, CAN uses Non-Return to Zero (NRZ) encoding, that is in case there are five consecutive

zeroes on the stream an extra '1' bit is added to that position before putting it on the bus.

A CAN Flexible Data rate (CANFD) frame has two additional bits in the control section. The first bit is the FDF, or the flexible data rate (*FD) Frame* bit. Setting it to high signifies that the current data frame is an FD frame. The second additional

SOF	Arbitration	Control	Data	CRC Field	ACK	End of Frame
1 bit	12 bits	6 bits	0 – 64 bits	16 bits	2 bits	7 bits

Figure 2: Standard Classical CAN Bus Frame.

bit is the Bit Rate Switch (BRS) bit. If this bit is set to high, it means that following data bits and the CRC bits are being sent at the flexible data rate. CANFD allows transmission of up to 64 bytes of data in a single frame. That is equal to eight frames worth of data of a classical CAN frame.

B. *Execution Isolation with Trusted Execution Environment*

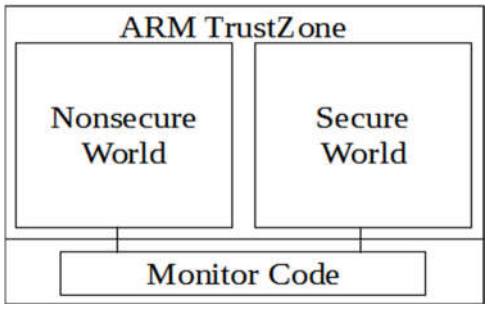

Figure 3: Birds Eye View of ARM TrustZone.

Traditional software based access control mechanism for isolation of sensitive processes and data, such as key generation and distribution mechanism are not cost effective in terms of speed and area. Software based techniques require a large overhead for implementation on an embedded device and are slow. Using the hardware based isolation extensions; key management processes and sensitive data can be kept separate through the use of hardware isolation and access privileges.

Trusted Execution Environment [2] provides hardware based isolated area for executing sensitive code. This mode is defined in the hardware and can be programmed through software. ARM provides an implementation of a trusted execution environment using TrustZone. TrustZone provides code isolation as well as isolation for on-board memory and peripherals. TrustZone divides the execution model of the processor into two contexts, the *Normal World* and the *Secure World*. The processor performs context switching between the two zones using a Secure Monitor Call (SMC) and Monitor Code.

1) Normal World

Normal world applications are non-secure and may require interaction with the user. These applications have a limited view of the memory depending on how the TrustZone restrictions have been set up. The code that runs in this world is considered to be insecure, and should also be considered to have security vulnerabilities.

2) Secure World

Secure world is processor context mode which executes security critical code. Secure world is set up in an isolated memory area which cannot be accessed by the normal world. This memory area and other security restrictions are set up with TrustZone after boot up. The secure world contains restricted algorithm, database or access to an on-board peripheral that normal world application should not have open access to.

3) Monitor Code and Secure Monitor Code (SMC)

Secure Monitor Call or SMC is a system call that is used for context switching between the two processor modes. When the SMC call is issued, an interrupt occurs. The system Interrupt Vector Table has an entry associated with the SMC. This code is called the *Monitor Code*. The monitor code sets up the alternate world before the context switch.

C. *Security flaws Associated with CAN Bus Communication*

CAN Bus is susceptible to replay attack, man in the middle and stealing identifiers attacks. We demonstrate these attacks using our set up consisting of Pi 2, CANBus Shield, and Arduino UNO.

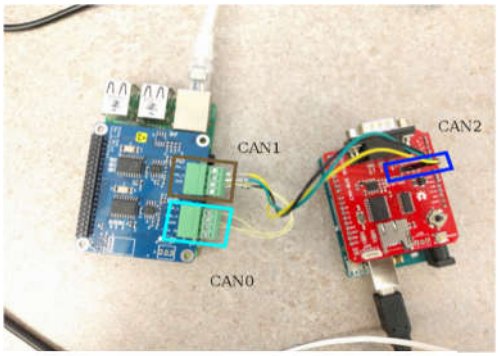

Figure 4: Experimental Setup

1) Test Bed Setup

The PiCAN shield has two CAN nodes, they are considered legitimate/trusted in our experiment, and are connected with the Raspberry Pi. These nodes are referred as CAN0 and CAN1, as shown in Figure 4. The malicious node, CAN shield is connected to Arduino UNO, refereed as CAN2,.

2) Threat Model

a) Eavesdropping

Figure 5 demonstrates that messages and destination identifiers are broadcasted and sent unencrypted on CAN. CAN0 messages for random nodes can all be seen and recorded by CAN1. Similarly, a device physically connected to the CAN bus can read all the communication over the network.

To mitigate eavesdropping, the information passing on the bus needs to be securely encrypted. In [3], the authors present an external security solution, and [4] proposes a 3DES encryption and controller for data transfers and traffic monitoring. However, added area, power requirements and slower performance of software based encryption and decryption processes make the solution impractical.

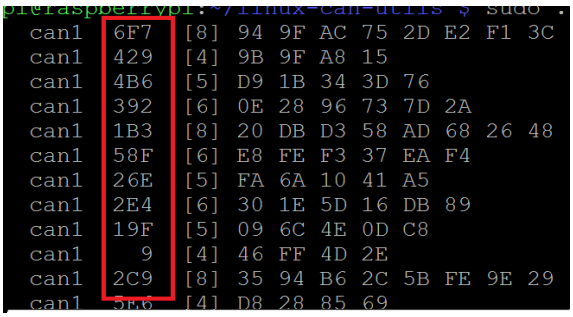

Figure 5: Screenshot of data transfer on CAN.

b) Stealing Identifiers

As shown in the Figure 5, all the identifiers are visible to the hacker/ attacker on CAN bus. All devices probe the ID field of each message received and based on the ID, deduce if the message was meant for them or not. Malicious device records all these identifiers and can launch attacks. In [5], an authentication and encryption mechanism is described, but it requires about three times more CPU cycles than the cycles required for normal operation of a CAN node.

III. Hardware Based Encryption and Digital Signature Solution

The limited processing power available on the hardware and real time constraints bound the traditional software based cryptographic solutions from providing secure communication. In embedded systems with a software solution, an attacker can extract the executable source code of the device using various methods to reverse engineer cryptographic algorithms and keys[6][7][8].

We propose hardware based secure enhancements to each connected electronic control unit (ECU). The secure framework can be embedded into the processor fabric itself as a co-processor. The hardware design is made to minimize software dependency on the resource scarce devices. Additionally, with incorporation of hardware based obfuscation techniques, reverse engineering attacks or attempts for extraction of keys using hardware based attacks can be further mitigated. The proposed framework requires no secret or shared keys to be stored on non-volatile memory within the nodes/ECUs, thus eliminating probing attacks.

The security framework is based on client server architecture. A local server node present on the network is responsible for handling all ECU nodes that are joined on the network. Each client must first be registered and authenticated at the server before they can communicate with the other nodes on a network. This process takes place in a secure and trusted environment, referred to as enrollment. Enrollment can be performed during manufacturing, assembly of vehicle components or at the trusted diagnostic centers.

Figure 6 shows the enhancements and design flow in form of block diagram of each secured ECU. The server is considered secure and trusted. Following subsections discuss each of the components and their function in the framework.

A. PUF Block

Physical unclonable function (PUF) is an emerging physical layer cryptographic primitive used in hardware security and privacy protocols. These are embedded structures that utilize

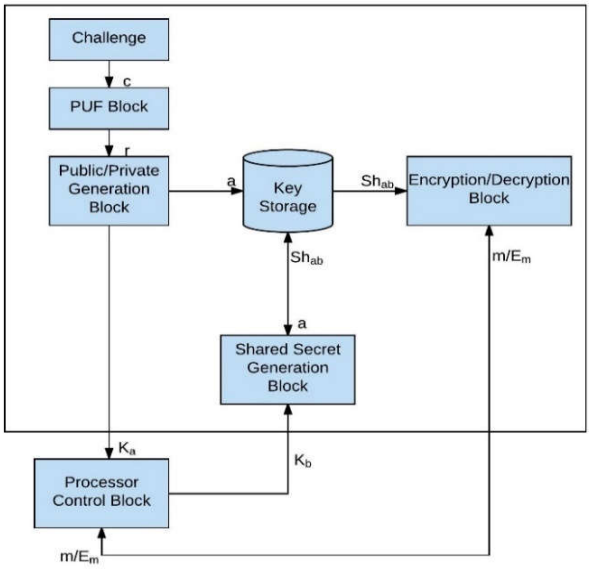

Figure 6: Secure block implemented at each client node.

inherent manufacturing process variations to extract unique but reproducible secrets [9][10]. PUFs are based on a challenge-response pair (CRP) mechanism. The challenge for a PUF is defined as a digital input, usually in the form of a bitstring of '0's and '1's. The output of a PUF is also digital but for most PUFs, this requires an on-chip mechanism to convert the small analog variations leveraged by the PUF to be digitized.

1) Hardware Embedded DeLay PUF (HELP)

Hardware embedded delay PUF (HELP) attaches to an on-chip functional unit, such as a portion of the Advanced Encryption Standard's (AES) sbox as shown on the left side of figure 7. The logic gate structure of the functional unit defines a complex interconnected network of wires and transistors, which includes 64 primary inputs (PIs) and 64 primary outputs (POs) and is implemented on the Xilinx Zedboard FPGAs using approx. 2,900 LUTs and 30,000 wire segments. Path delay is defined as the amount of time (Δt) it takes for a set of 0-to-1 and 1-to-0 bit transitions introduced on the PIs of the functional unit (input challenge) to propagate through the logic gate network and emerge on a PO. HELP uses a clock-strobing technique to obtain high resolution measurements of path delays [11]. The digitized path delays (referred to as PNs) are collected by a storage module and stored in an on-chip block RAM (BRAM), as shown in the center of figure 7. Once a set of 4096 PNs are collected, a sequence of operations implemented in an VHDL are started to produce the bitstring and helper data, as shown on the far right of figure 7.

The high cryptographic quality of the HELP PUF builds a solid foundation for reproducing reliable, unique and unpredictable response used for the Public/Private key Generation Block.

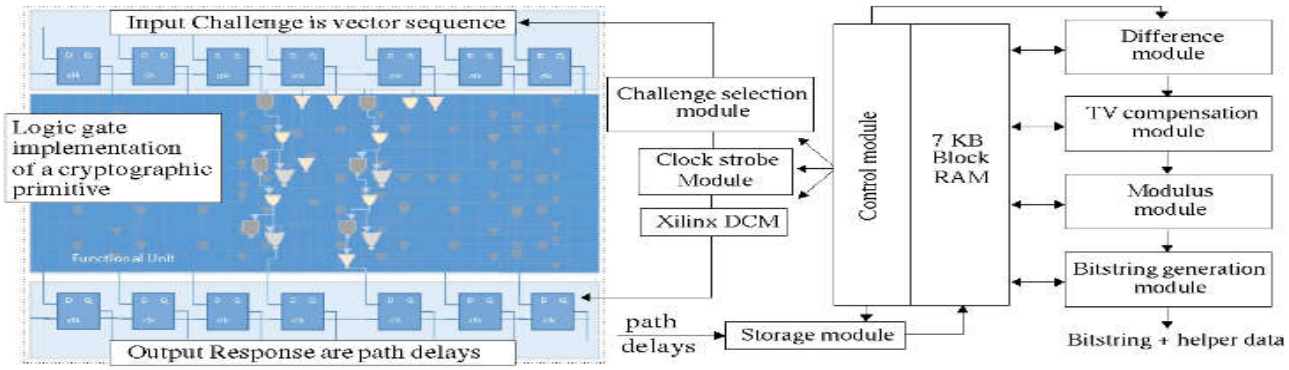

Figure 7: HEL PUF

2) Bit-generation Process

During the enrollment process, each registered ECU is challenged with a seed and configuration parameters. Each ECU produces a unique reproducible response/key that serves as private key. Using this response and stored configuration parameters as inputs to the Private / Public Key Generation Block (explained in the proceeding subsection) public key is generated. Helper data is stored on the NVM of ECU to regenerate same response for a given challenge and public key is communicated to the secure server, where it is stored. PUF based authentication is implemented within the vehicle network [10]. All client nodes (or devices) are enrolled at the server before they can be used. The enrollment process is summarized in Table I.

TABLE I. ENROLLMENT OF NODES IN A SECURE ENVIRONMENT.

Algorithm: Enrollment process of client nodes in a trusted environment.
Input: Challenge c, configuration parameters for the ECC curve.
Output: Public key K_a of ECU.
1: At each client node, input a challenge *c* for the PUF Block.
2: PUF block computes a response *r*.
3: Generate a public private key pair (a, K_a) using PUF response r and configuration parameters.
4: K_a for each ECU is stored communicated to server, where it is stored in a public database accessible to every node on the bus.
5: Public key of the server, K_s is stored in non-volatile memory at each client node for later access.

B. Private / Public Key Generation Block

We implement Elliptic Curve Cryptography Diffie-Hellman (ECDH) based asymmetric key exchange and encryption engine in hardware. ECC algorithm is suitable for the resource constraint devices, providing same strength of security as RSA. ECC algorithms are implemented in Galois Field(GF) which limit the range of output values and thus reducing the resource requirements. ECC implementations have been presented over two types of Galois Fields, namely Prime Fields (GF(Fp)) and Binary Fields (GF(F2^m)). National Institute of Standards and Technology (NIST) provides a set of recommended elliptic curves [12].

ECC algorithm uses the PUF response as private key and generates public keys for a node. It is used in the one-time enrollment process. Based on the reliable PUF response, public and private keys are generated on each ECU during the authentication process that is every time the vehicle starts (turning on of the ignition).

TABLE II. ALGORITHM FOR AUTHENTICATION PERFORMED AT THE BEGINNING OF EACH SESSION.

Algorithm: Authentication process.
Input: None.
Output: None.
1: Each ECU generates PUF response r using stored challenge c, and helper data.
2: With r as the input to the Public / Private Key Generator Block, generate public and private keys, Ka and a respectively.
3: With public key of the server Ks and private key a, generate shared symmetric encryption key Shab using the Shared Key Generation Block.
4: Encrypt the generated public key Ka with the shared key Shab to form message Em.
5: Transmit Em to the server while setting the node ID IA of the current node as CAN message arbitration ID.
6: At the server, once an encrypted message Em is received, Server decrypts it using the shared encryption key associated with A to get message m, which holds the public key of node A, Ka.
7: If Ka received is equal to the Ka stored in the database then consider node A authorized for the rest of the session.
8: Wait for other nodes to register themselves.
9: Server prepares a message for each ECU encrypting with the ECU's shared key containing the public keys of all the registered nodes.
10: Once this message is received at a node, decrypt it using the shared key.
11: Each node communicates with the other nodes and generates session keys using both its private key and other nodes public key.

C. Authentication and Shared Key exchange Process

During the startup, each connected device requires authentication with the server to enable trusted environment. ECU sends a packet with its own ID as target identifier and its public key encrypted with the shared key formed using ECC in the payload. Server decrypts the payload and compares the public key with the sender ID sent in the arbitration bits. Once the ECU is authenticated, server shares the shared secret keys of the other ECUs the requesting node is allowed to communicate. The authentication algorithm is formulated in Table II. The authentication process is a timed process. If any node fails to authenticate itself with the server in the time allocated, then it is blacklisted for communication for the entirety of that session, and the user dashboard is set to display the alert. Private keys for all nodes are generated at run time and are not stored to ensure security. No private keys leave the node and are erased at the

end of session. Figure 8 illustrates the sequence of communication during authentication of a node.

For the prototype of our framework, we have selected an ECC engine operating over the binary field GF (2^163) [13]. One Finite Field Multiplication operation takes 7.14 µs for a

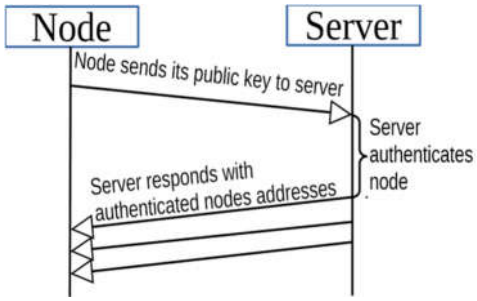

Figure 8: Sequence Diagram for Authentication.

clock speed of 200 MHz.

D. Key Storage

The key storage block consists of a non-volatile memory (NVM) to store:

- Seed, challenge and helper data to generate unique, random and reproducible PUF responses.
- Public key Ks of the server for server authentication.
- Parameters to generate public private key pairs.

The framework allows each ECU to generate the private key on fly thus does not require it to be stored in a NVM. Private keys can be stored temporarily on a volatile memory along with the shared session keys between ECUs. For each pair of communicating nodes, the following elements are stored in volatile key storage and are isolated from the untrusted applications and CAN bus interface using TrustZone:

- Sh_{ab}: Generated Shared secret key for encryption for each pair of nodes (a being the current node and b the node being communicated).
- K_b: Public key of the communicating node.

No access is possible to the storage other than to the nodes connected to it. For implementing the storage elements of the framework, we are using the FPGA's on-fabric block RAM resources. A block RAM of 128-bit width and 512 elements is instantiated. On our target board, a total of two 36K BRAM elements were used. This memory element has a read and write speed of one cycle.

E. Encryption / Decryption Block

Once shared keys are generated for each pair of nodes, these keys can be used for encryption and decryption for the given session. Every message that is sent must first be encrypted before it can be transmitted. Selection of a symmetric key algorithm is dependent on the choice of the system designer, for example an AES-128 or AES-256 engine. The message is sent to the Encryption / Decryption Block that encrypts the message at the sender and decrypts the message at the receiving node using the shared key. Our reference framework uses AES-128 for encryption. The AES engine uses the shared key generated by the ECC subsystem to encrypt any traffic leaving the node and decrypt any traffic coming in. AES-128 engine takes a total of 22 clock cycles for encryption and decryption and a total of 110 nanoseconds. The AES engine has a footprint of 2560 LUT slices.

F. Hardware Access Isolation

To maintain isolation between the ECU processes and the secure cryptographic IPs and key exchange, we implement TrustZone. The secure framework can only be accessed through the secure world. To provide an interface to the ECU code, we have developed 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 III.

TABLE III. SECURE ZONE API

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 a 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 particular 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.

IV. PERFORMANCE ANALYSIS

The overall timing overhead of the proposed framework with system clock speed of 200MHz, CAN speed as 1Mbps and CANFD speed for 8Mbps is discussed. Time for transmitting one bit at the data rate of 1Mbps is 1µs whereas for 8MBps is 125ns. We are ignoring bit stuffing due to NRZ encoding in our computation. Since for sending a CANFD frame, the frame excluding the data bits and the CRC are sent on the classical CAN frame speed, this amounts to a total of 30 bits. To send these bits on a 1Mbps connection, it takes a total of 30 µs.

1) During Authentication

At the time of boot up, each node generates a public key and private key pair. This process requires one point multiplication operation to be performed in the ECC core. This operation requires a total of 7.14 µsecs. Using the stored public key of the server, another point multiplication operation will be required to generate the shared encryption key. As per Table II, this key is encrypted before it is sent to the server. The encryption operation takes 110 ns. Therefore, the total minimum time required from boot-up to sending the server the shared key is:

$$(7.14\mu s \times 2) + 110ns + (30\ \mu s + (64+16) \times 125ns) = 24.42\mu s.$$

For each node that has been authenticated by the server, the server will send a two-part message to all the connected nodes. The first part contains the node ID of the node authenticated, and the second part consists of the encrypted public key of an authenticated node to establish shared key for inter ECU communication. Time taken to send these messages is:

$$30\mu s + (11+128) \times 125ns = 47.375\ \mu s.$$

This process is repeated for all successfully authenticated nodes. The public key of an authorized node is encrypted in the message, the decryption process takes another 110 nanoseconds. To generate the shared key for that public key requires another 7.14 µs by the ECC unit.

2) During Normal Operation

During normal operation, a single CANFD message needs to be encrypted, a frame of CANFD message allows upto 64 bytes, thus requires a single frame to transmit the message.

TABLE IV. OVERHEAD OVERVIEW AT STANDARD CAN CONNECTION SPEEDS WITH CANFD SPEED OF 8MBPS IN NORMAL OPERATION

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

Table IV gives a concise overview of the message overhead incurred for authentication and typical communication operation.

V. 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 bootup. 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.

VI. CONCLUSION

In this paper, we investigated 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. We implement the framework with hardware based authentication and point to point encrypted communication for ECUs. The paper provides an in-depth description, implementation and performance analysis in terms of timing analysis and area overhead. To conclude, we present a discussion on security analysis of the proposed framework.

VII. REFERENCES

[1] Standard I.S.O., "Iso 11898," *Road Veh. Digit. information–Controller Area Netw. high-speed Commun.*, 1993.

[2] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-TEE -- An Open Virtual Trusted Execution Environment," in *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 400–407.

[3] A. Hanacek and M. Sysel, "Design and Implementation of an Integrated System with Secure Encrypted Data Transmission," Springer International Publishing, 2016, pp. 217–224.

[4] G. Singh, "A study of encryption algorithms (RSA, DES, 3DES and AES) for information security," *Int. J. Comput. Appl.*, 2013.

[5] Q. Wang and S. Sawhney, "VeCure: A practical security framework to protect the CAN bus of vehicles," in *2014 International Conference on the Internet of Things (IOT)*, 2014, pp. 13–18.

[6] S. Ravi, P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual conference on Design automation - DAC '04*, 2004, p. 753.

[7] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, 2015, pp. 145–152.

[8] F. Garcia, D. Oswald, and T. Kasper, "Lock It and Still Lose It–On the (In) Security of Automotive Remote Keyless Entry Systems," in *USENIX Security 2016*, 2016.

[9] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical Unclonable Functions and Applications: A Tutorial," *Proc. IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug. 2014.

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

[11] 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.

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

[13] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko, "A high performance pseudo-multi-core ECC processor over GF(2 ^ 163)," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 701–704.