Efficient Crypto Engine for Authenticated Encryption, Data Traceability, and Replay Attack Detection Over CAN Bus Network

Amar Rasheed, Mohamed Baza, Mahmoud. M. Badr, Hani Alshahrani, and Kim-Kwang Raymond Choo, *Senior Member, IEEE*

***Abstract*—Smart vehicles and industrial control systems becoming increasingly complex. They are comprised of a large number of connected intelligence sensor devices. For such systems, Controller Area Network (CAN) bus offer high-integrity serial communication capabilities. It transformed the way how these systems are networked. Due to the lack of data security features on CAN-enabled systems, many of these systems are vulnerable to a wide range of cyber threats. This article proposed the development of a crypto-based subsystem that is capable of supporting CAN authenticated data encryption/decryption, crypto-provable data traceability, and replay attack detection capabilities. Data confidentiality was achieved via the deployment of a lightweight block cipher authenticated encryption scheme based on TinyJAMBU-128. Crypto-provable data traceability was accomplished through the utilization of a block-chaining approach. Meanwhile, an anti-replay attack mechanism that implements CAN message context awareness has been tested and validated under various data infection rates. Our CAN security subsystem was fully implemented and deployed on a testbed with multiple STM32 Nucleo development boards. System performance for our security schemes was analyzed and compared with traditional encryption schemes AES, ARIA, and Camellia with SHA-512 for supporting message authentication. Based on our performance results, the proposed security subsystem achieved the lowest CAN bus load and average message overhead compared to other encryption schemes. In the case of the anti-replay attack mechanism, we were able to reach a detection rate of 99.99% for data infection rate below 20%.**

***Index Terms*—Encryption, CAN Bus, authenticated encryption, decryption algorithm.**

Manuscript received 2 April 2023; revised 11 August 2023; accepted 30 August 2023. Date of publication 8 September 2023; date of current version 8 January 2024. Recommended for acceptance by Dr. Yang Xiao. *(Corresponding author: Mohamed Baza.)*

Amar Rasheed is with the Department of Computer Science, Sam Houston State University, Huntsville, TX 77340 USA (e-mail: axr249@shsu.edu).

Mohamed Baza is with the Department of Computer Science, College of Charleston, Charleston, SC 29424 USA (e-mail: bazam@cofc.edu).

Mahmoud. M. Badr is with the Department of Network and Computer Security, Suny Polytechnic Institute, Utica, NY 13502 USA (e-mail: badrm@sunypoly.edu).

Hani Alshahrani is with the Department of Computer Science, College of Computer Science and Information Systems, Najran University, Najran 61441, Saudi Arabia (e-mail: hmalshahrani@nu.edu.sa).

Kim-Kwang Raymond Choo is with the Department of Information System and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78260 USA (e-mail: raymond.choo@utsa.edu).

Digital Object Identifier 10.1109/TNSE.2023.3312545

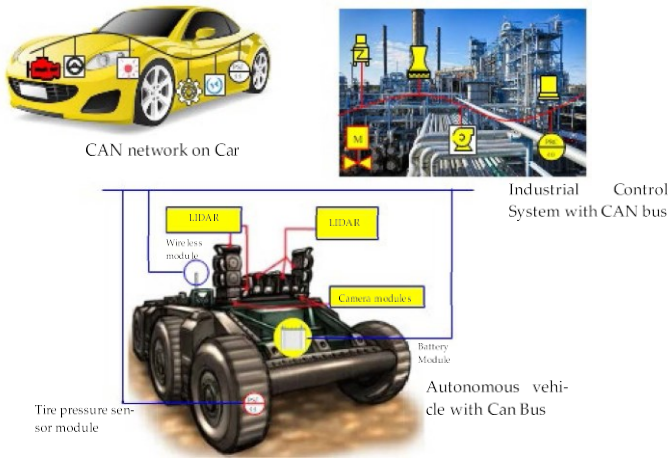

Fig. 1. CAN BUS applications.

I. Introduction

CONTROLLER area network (CAN) protocol has been widely embraced by the automotive industry (see Fig. 1). In today's connected vehicles, CAN enables internal components of the vehicle's complex system to communicate with one another without the need for a central processing unit. CAN bus provides low-cost and reliable data communication solutions. Data generated from various sensor units can be transmitted over the bus at the rate of 2 to 5 Mb/s. Existing vehicle systems can support more than 70 intelligent sensors and Electronic Control Units (ECU)s. The biggest processing unit in any vehicle system is the engine control unit. Other ECUs that support autonomous driving capabilities include Advanced Driver Assistant System (ADAS) unit, airbags, cruise control, battery and recharging system, and lane assist/collision avoidance.

Additional CAN features like flexibility, reliability, ruggedness, and high tolerances against interference makes the technology an ideal data communication platform for many safety-critical industrial control system, and autonomous military platforms (e.g., unmanned Aircraft Systems (UAS) and Lethal Autonomous Weapon Systems (LAWS)). Tactical military vehicles are highly integrated with multiple sensor fusion modules and control algorithms. In such systems, rapid decision-making, fast maneuvering, and collision avoidance capabilities are vital

for supporting mission dynamics. In both commercial and military-connected vehicles, CAN protocol shows superiority in the way how various sensor data get prioritized and transmitted over the CAN network. For example, camera, LIDAR (Light Detection and Ranging), GPS (Global Positioning System), engine temperature, and acceleration data computed by the various subsystems are prioritized based on their assigned CAN message IDs. Similarly, CAN bus integration into today's industrial control systems brings an essential functionality that ensures the safe operation of such systems. Data prioritization capability for sensor data and actuators is inherently supported by CAN which eliminates the need for additional hardware or software components.

Although CAN bus technology has been incorporated into the design of a wide range of interconnected systems, traditional CAN systems still lack the support of basic security primitives [1], [2]. Data encryption/decryption, message authentication, and data integrity verification capabilities are not integrated into current CAN technologies. CAN is a message-based broadcast system, in which messages can be easily extracted and compromised from the bus via the deployment of a malicious ECU acting as a packet sniffer. Non-authenticated CAN data messages can be altered during transit and injected into the network. CAN protocol offers zero resiliency against message replay attacks. Captured CAN message from previous data transmissions can be inserted into the CAN bus via a compromised ECU. Furthermore, current CAN technologies are inadequate in providing cryptographically secured data tracing capability. Specifically, in the case of a car accident, non-authenticated CAN messages transmitted over the bus that holds critical information such as acceleration data, tire pressures, and steering wheel angles are not cryptographically secured. A malicious ECU will be able to alter the sequence of data in the trace by placing additional packets into the trace or even changing the order of the packets within the trace.

The main thrust of this research includes the development and deployment of an efficient CAN security subsystem for CAN-based interconnected systems. The proposed CAN security subsystem supports two auxiliary security modules integrated into the existing TinyJAMBU-128 [32] authenticated encryption scheme. The newly developed security modules enhanced TinyJAMBU-128's security capabilities against CAN replay attacks. The modified TinyJAMBU-128 is capable of achieving secure data traceability and data authentication over CAN bus networks.

Finally, systematical evaluation and third-party methodological frameworks that assess the security property of CAN-based encryption systems have not been proposed in the past. Existing research literature mainly focuses on classical crypto techniques that were designed for general-purpose computing systems [44], [45], [46]. Also, due to CAN bus system requirement, many of today's performance metrics that are used to analyze the efficiency of traditional security mechanisms might be unsuitable for evaluating security algorithms developed for CAN bus network. We have investigated the employment of two performance metrics that fully describe the impact of deploying our proposed crypto system: (i) CAN bus overhead capacity and (ii) ECU processing power. Experimental results that capture the modified TinyJAMBU-128 authenticated encryption system efficiency in terms of CAN bus overhead capacity and ECU processing time were presented during this work.

The following cryptographic techniques have been developed and deployed onto the proposed CAN security subsystem.

- *Lightweight Authenticated Data Encryption Scheme:* Efficient and lightweight encryption/decryption scheme based on TinyJAMBU-128 [32] authenticated encryption was implemented. We have evaluated the performance of the proposed authenticated encryption scheme and compared it with traditional crypto engines that support block cipher encryption. Performance metrics based on CAN bus load, message overhead, and processing time were captured for TinyJAMBU-128 [32], AES128, AES-192, AES-256 [44], SHA-256, ARIA-128, ARIA-192, ARIA-256 [45], Camellia-128, Camellia-192, and Camellia-256 [46]. Data encrypted via AES, ARIA, and Camellia were authenticated via the deployment of SHA-512. Based on our simulation results, the proposed scheme achieved better performance in terms of message overhead, and CAN bus load compared to other encryption schemes. We have analyzed the performance of the proposed replay attack detection scheme based on true positive and false positive rates under various malicious data injection rates.
- *Efficient Countermeasure Mechanism Against Replay Attacks*: A cryptographic-based approach that utilizes CAN channel data communication patterns to derive a transmission context for each data message sent over the bus. In our proposed scheme, transmission contexts are encoded into the associated data packets that are used for encrypting CAN messages. Our proposed scheme can achieve a 99.9% replay attack detection rate. CAN messages transmitted within an invalid context will be detected. We have tested the proposed scheme by injecting replay messages into the CAN bus using various injection rates. We have validated the ability of the system for identifying malicious data packets sent within invalid transmission contexts.
- *Crypto-provable Technique for Data Traceability:* To ensure the integrity of a trace, CAN data transmitted over the bus are block-chained using a cryptographic hash approach. Our proposed scheme provides full resiliency against data modification and data injection attacks. A compromised ECU will not be able to insert malicious CAN data into a pre-computed data-trace.

Our proposed techniques are capable of supporting secure data transmission, message authentication, and crypto-provable data traceability function. Data communicated over the CAN bus will be encrypted and authenticated with minimum data communication overheads. Our CAN security subsystem makes existing CAN bus-based interconnected systems fully immunized against replay attacks and threats that targeted the integrity of the data-trace.

The proposed CAN subsystem was implemented on multiple STM32F411 [47] boards. Each STM32F411 board features an ARM Cortex-M4 processing unit, 512KB of flash memory, and 128KB of SRAM, and it is CAN-enabled. We have created

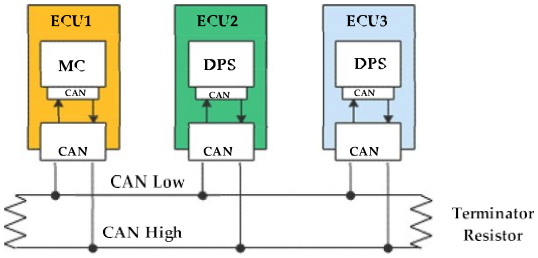

Fig. 2. CAN bus architecture.

a CAN bus-based interconnected system where STM32F411 systems send and receive data over the CAN bus. Data confidentiality, message authentication, and system resiliency against replay attacks were tested using the added CAN security subsystem. Our security subsystem shows superiority in achieving a high detection rate against replay attacks and is capable of reinforcing secure data transmission with minimal communication overheads.

This article is organized as follows. Section II presents the CAN protocol. Section III describes related works. Section IV introduces a high-level architecture of the proposed CAN security subsystem. Section V introduces a lightweight encryption/decryption algorithm based on the TinyJAMBU-128 crypto engine. Section VI describes the proposed countermeasure technique against replay threats. Section VII presents a cryptographic technique based on hash for supporting the integrity verification of data-trace. Section VIII introduced the security analysis approach. Section IX shows the performance results of the proposed CAN security subsystem. Section X illustrates the interconnected system architecture for the proposed testbed. Section XI concludes the article and presents future work directions.

II. Backgrounds

CAN was first introduced and developed by BOSCH [3] and is a message-based protocol that supports the broadcasting of CAN data where there is no centralized unit that facilitates how data get transmitted over the bus. The original CAN was able to provide a maximum data communication rate of 1Mbit/sec. CAN feature a point-to-point communication system, data is transmitted as small blocks of size 1-8 bytes per message. For more than thirty years, CAN provided flexibility and upgradability to the automotive industry where complex wiring systems were replaced by CAN-High and CAN-Low wire systems (see Fig. 2). New ECUs can be added to the network with minimal modification to the internal system architecture. ISO-11898 defined how CAN protocol is mapped to the ISO Data-link layer and Physical layer. In this research, the CAN security subsystem was fully implanted and integrated into the application layer, while preserving the CAN protocol's original design integrity. Our subsystem will serve as plug and play system, it can be initiated at ECU connected to the systems when data security feature is imperative for ensuring the safe operation of the system.

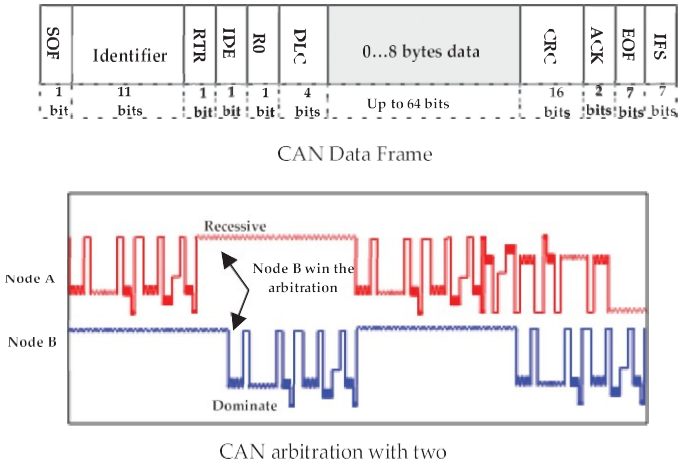

Fig. 3. CAN data frame and CAN arbitration.

One important aspect of CAN communication is that it relies on carrier sense, multiple access mechanisms with collision detection plus arbitration on message priority (CSMA/CD+AMP) for accessing the bus. In CAN, ECUs contention over the bus is resolved using a bit-wise arbitration method, messages are prioritized based on their programmable ids, and messages with the highest priority will win the arbitration race. The original ISO-11898 support a data communication rate from 125kbit/sec to 1Mbit/sec with an 11-bits message id. The new CAN is able to provide an extended 29-bits id where 2^{29} different identities can be created.

A. Message Arbitration

CAN bus was implemented using a twisted and shielded wiring system with 120 Ohm termination resistors on each side of the bus. CAN is based on a differential voltage signal transmitted between CANH and CANL. When the bus is idle, it remains in a recessive state until one node pulls the state of the bus to dominate. When two nodes try to send data over the bus simultaneously, their messages can be corrupted or destroyed. CAN resolves bus contention via the bit-wise arbitration approach. Each message transmitted on CAN is identified by an 11-bit ID or 29-bits in the case of standard CAN and extended CAN respectively. Message prioritization is computed based on the message's identifier bit values. Messages with lower binary values in their identifier fields have higher priority. For example, a CAN message with its identifier bits field set to dominate (0) will have the highest priority. When two nodes compete for the bus, a node with its last transmitted identifier bit set to dominate will win the arbitration. While the winning node continues in its frame transmission, the second node stops its data transmission. The arbitration process is illustrated in Fig. 3.

B. Data Types

There are four types of CAN messages transmitted over the bus, error frame, overload frame, remote frame, and data frame.

- *Data Frame*. This message contains the CAN payload, message identifier field (11 bits for standard CAN and 29 bits for extended CAN), the CRC data, and 2-bit ACK.
- *Error Frame*. Corrupted CAN messages are detected by computing CRC on the received data packet. When a distorted CAN frame is identified, all receiving nodes start sending data error frames over the bus. CAN controller implements an error counter to prevent a transmitter from obtaining exclusive possession of the bus by repeatedly transmitting error frames.
- *Remote Frame*. A remote frame is used to request data from another node connected to the bus. The remote frame has the RTR bit set to a recessive state. Every node on the bus will receive the remote frame message, but only nodes that are interested in the data will send their replies to the transmitter.
- *Overload Frame*. A mechanism that produces delays between CAN messages is required to stabilize the state of the bus, especially when nodes become too busy and unable to process all incoming data.
- *Valid Frame*. An error-free Data frame will have the last bit in the EOF field set to recessive. While messages with EOF bit sets to dominate, CAN mark it as an error in the message which require retransmission.

III. Related Works

Recent works on threat mitigation and intrusion detection systems [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [50], [51] show promising results in detecting CAN bus attacks. Attacks [14], [15] that target the data integrity of safety-critical sensor system in a modern vehicle, such as the vehicle Anti-lock Braking System (ABS) has been presented in [22]. Liuwang K. [22] proposed a threat detection and attack mitigation technique against ABS CAN bus attacks. Their method is capable of identifying sensor attacks and CAN bus attacks related to ABS with the execution of an attack mitigation strategy that enables dynamic threat isolation. The authors proposed the development of a system that predicts the current state of the vehicle based on measured historical vehicle state data, derived from multiple sensor measurements. In their approach, the vehicle state space is comprised of real-time road friction data (i.e., wheel speed and longitude break force). Threat mitigation was implemented based on the subtraction of anomalous data from the measured vehicle state data to establish the correct reading for the vehicle ABS. One shortcoming of the above approach, hardware integrity verification of ECUs was not considered during vehicle state identification. Vehicle state ABS computed during normal operations might contain false data, which leads to false predictions. For example, during the vehicle state identification process, malicious ECUs attached to the bus can inject anomalous data altering the vehicle state data. To overcome this problem a full inspection needed to be performed before incorporating the vehicle state into the prediction model.

Data injection based on replay attacks has been exploited in [23]. P. Thirumavalavasethurayar and T. Ravi [23] proposed the implantation of attacking the CAN bus by injecting replay messages over the bus. A testbed of three CAN-enable nodes was implemented using a universal verification methodology. A malicious node was deployed to simulate replay attacks. Two classes of replay attacks were developed full and partial frame replaying attacks.

Existing works based on sequence-based detection algorithms [24], hidden Markov models [25], and neural network approaches have shown promising results in detecting message injection and replay attacks on the CAN bus. The work presented by Satya Katragadda [26] illustrated the effectiveness of the sequence-based anomaly detection algorithm in detecting low-rate replay threats for over 99% f-score. The proposed scheme achieved better performance compared to existing dictionary-based algorithms and a multi-variate Markov chain-based approach. Mubark Jedh [27] presented a novel approach based on similarities of successive messages-sequence graphs for detecting Message injection attacks. A detection algorithm based on generating a Messages-Sequence Graph (MSG) that presents CAN messages as sequences of data sent within a given time interval. In their work, the detection of message injection attacks was achieved through the deployment of cosine similarity and Pearson correlation methods. Sequences of MSGs were used to compute similarities that might exist in successive MSGs, enforced by change point detection, and Long Short-Term Memory (LSTM) to predict injection attacks on the CAN bus. The proposed scheme was able to sustain a detection rate of 98.45% and 1.5 to 2.64 response time. Techniques that rely on a machine learning approach or sequence-based intrusion detection system usually require heavy processing and large storage capabilities to train and estimate the model parameters. Real-time model training and parameter estimation are not visible with current CAN bus technology due to CAN nodes' limited processing power and storage capacity. To overcome this problem, two approaches in model training were considered in the literature. The first approach is based on offline training techniques, where CAN data is extracted first and then feed into a training algorithm running on a high-performance computing system. The second approach implements real-time feature extraction algorithms and extracted CAN data is communicated directly to a training model running on the cloud.

Security approaches based on message authentication [16], [17], [18], [19], [20], [21], hash-based message authentication code (HMAC) attribute-based encryption, symmetric-based encryption techniques, and ECC-based key management algorithms have been investigated in [28], [29]. A crypto-based technique for detecting CAN message injection and replay attacks was presented by the work of Timothy Dee in [30]. It enhances existing CAN-FD technology by incorporating message integrity, message authentication, and source node authenticity capabilities. Message freshness was implemented through the maintenance of freshness value tables. Security algorithms implemented with classical authentication schemes require heavy processing and introduce large communication overhead to the CAN. A standard CAN data frame is limited to a 64-bit block of data. Therefore, authentication data or an encrypted message with a block size larger than 64 bits will require more than one CAN frame to transmit the data over the bus.

Several lightweight detection algorithms against replay attacks have been proposed in the past. F. Páez and H. Kaschel proposed an algorithm that [31] eliminates the need for employing computationally intensive crypto-based methods for detecting CAN threats. It relies on the incorporation of a new CAN ID into the messages' acceptance filter of nodes. The proposed approach was tested using a CAN-enable testbed. Modified and replayed CAN messages were identified by the proposed method at 40μs detection speed.

Recent work that analyzes the vulnerabilities of attribute-based signature schemes has been presented by Zhaozhe Kang and Jiguo Li [33]. The authors show that existing attribute-based signature scheme can be exploited by attackers through signature abusing and key exposure. To overcome these issues, the authors proposed a novel scheme that supports traceable and forward-secure attribute-based signatures (TFS-ABS). Their proposed scheme has been proven for enabling unforgeability features against selective predict e attacks for the standard model. Furthermore, based on their simulation results, the proposed TFS-ABS was capable of achieving efficient communication and computation overhead.

Public key authenticated encryption with keyword search (PAEKS) has been presented by Yang Lu and Jiguo in [34]. The authors show that data security schemes based on searchable public key encryption (SPKE) techniques are vulnerable to keyword-guessing attacks. PAEKS scheme has been proven to show resiliency against such attacks. To improve PAEKS resiliency against adaptively-chosen targets adversaries, adversary model of PAEKS has been refined and modified. Finally, a light weight PAEKS scheme that minimizes the computation power of performing bilinear pairing operations has been implemented. It showed that the improved light weight PAEKS was capable of supporting low communication and computation profiles, which makes the scheme more suitable for power-constrained devices.

Joseph Bonneau and Cormac Herley [35] examine the difficulty of replacing passwords in today's web-based user authentication schemes. In their study, the authors showed that there is a wide range of security approaches that offered password replacement solutions. Such schemes were capable of providing extra security primitives that extend existing password capabilities. But, with the extra security benefits comes a system deployment cost and usage which make these security systems less attractive compared to password-based user authentication techniques. Finally, the authors provide an evaluation methodology and benchmark for validating and testing future web authentication schemes.

Ding Wang and Ping Wang [36] presented a comprehensive analysis of how today's two-factor authentication schemes and smart-card-based password authentication mechanisms are being poorly evaluated and assessed. A fully comprehensive security assessment model and benchmarking for analyzing important security features of the proposed schemes become indispensable tools. The authors proposed a security model that is capable of extracting import features of an adversary and generating a set of twelve properties for system testing. Their main contribution is to devise a new security approach that offers full resiliency against user corruption and server compromise threat models.

Another important study expresses the need for a full comprehensive systematical assessment methodology that authentication scheme designers will be able to use to assess their proposed schemes. Ding Wang, Wenting Li, and Ping Wang [37] showed that the lack of comprehensive assessment tools leads to what they called a "break-fix-break-fix" cycle in the area of two-factor authentication schemes for securing data in industrial wireless sensor networks (WSNs). In their work, 44 schemes were tested under their proposed evaluation framework. The proposed evaluation framework provides unrepresented evaluation metrics for two-factor authentication schemes in industrial WSNs.

In the area of autonomous vehicles (Avs) [38], Qi Jiang and Ning Zhang proposed a cloud-centric three-factor authentication and key agreement protocol (CT-AKA). The authors illustrated how AV with a control capability poses potential threats to passenger safety. As the system could be exploited by an attacker and therefore gain him/her full access to the AV system remotely. The proposed CT-AKA was integrated with passwords, biometrics, and smart card capabilities. To achieve three-factor authentication, CT-AKA was implemented with three biometric encryption approaches, including fuzzy vault, fuzzy commitment, and fuzzy extractor. To test the visibility of CT-AKA, security properties were evaluated, and simulation results showed that their proposed approach was capable of achieving high security with acceptable communication computation overheads.

A secure user authentication scheme for cloud-assisted IoT systems has been proposed by Chenyu Wang and Ding Wang [39]. The authors demonstrated the requirement for a lightweight user authentication protocol to ensure secure access to IoT data over the cloud. They were able to analyze IEEE TDSC 2020 scheme to identify common vulnerabilities and challenges for designing an efficient light weight cloud-assisted user authentication scheme. Security analysis based on the random-oracle model, heuristic approach, the ProVerif tool, and BAN logic were used to assess their proposed scheme. Based on a predefined list of security requirements, their proposed scheme was able to achieve minimum computation and storage overheads on the gateway.

Qingxuan Wang and Ding Wang [40] discussed the visibility of attacking smart-card-based password authentication mechanisms via quantum computing. With the vast amount of processing power available through quantum computing systems, keeping the current two-factor authentication system unexploitable poses a great security challenge for systems designers. The authors presented the design of a secure and efficient smart-card-based password authentication scheme. Their newly proposed scheme called "quantum2FA" employs Alkim et al.'s lattice-based key exchange and Wang-Wang's "fuzzy-verifier + honeywords" approach (IEEE TDSC'18). The scheme offers resiliency against the revealed key-reuse attack against a lattice-based key exchange. Security analysis based on the random oracle model has been examined to assess the security properties of "quantum2FA". Their experimental results show that quantum2FA offers better computation speed as compared to existing 2FA techniques.

In this article, we present the implantation and deployment of a lightweight power-aware crypto engine that support authenticated encryption based on TinyJAMBU-128 [32], crypto-provable data traceability, and intrusion detection capability against replay attacks. The proposed engine minimizes communication overhead by fitting authentication data or encrypted data into a single CAN frame. Our proposed engine enables message authentication by adding one additional CAN frame to each CAN data transmitted.

IV. The Proposed Security Subsystem

We proposed a CAN security subsystem that is capable of supporting multiple security features to the existing CAN bus. Crypto-based security blocks that have been implemented and deployed include (i) a lightweight authenticated encryption engine based on TinyJAMBU-128 [32]. Data transmitted within a CAN data frame were encrypted and authenticated using a modified version of TinyJAMBU-128. (ii) lightweight detection algorithm against CAN replay attacks (iii) block chaining based on hash computation algorithm that enables the secured capture of data traces. Each CAN data is encoded into a single hash block where each block is constructed by computing a hash value $H(D_i \parallel H(bl_{i-1}) \parallel TID_i \parallel T_i)$, where D_i is CAN data field, $H(bl_{i-1})$ hash of the previous block, TID_i is the transaction id for block i, and T_i is data transmission timestamp. A detail description of symbols used throughout the article is presented in Table I.

TABLE I
Description of Symbols

Symbol	Description
$H()$	Hash function
$H(bl_{i-1})$	Hash of the previous block bl_{i-1}
D_i	The i-th CAN data field
TID_i	Transaction id of the i-th block
T_i	Data transmission timestamp
K	128-bit TinyJAMU encryption key
k_j	The j-th key bit
$\mathcal{S}$	128-bit TinyJAMU internal state
s_j	The j-th state bit
$\oplus$	Bitwise exclusive or operation
$\parallel$	Message concatenation
ID_i	11-bit id for the i-th CAN data frame
id_j	j-th bit of CAN data frame's ID
τ_{M_i}	64-bit authentication tag string for CAN message M_i
τ_j	The j-th bit's authentication tag
τ'_{M_i}	64-bit recomputed authentication tag at the receiving end for CAN message M_i
$\tau_j{}'$	The j-th bit of the recomputed authentication tag
Ad_i	64-bit associated data string for the i-th CAN data frame
M	64-bit plaintext CAN message
m_j	The j-th bit of the plaintext message
C	64-bit ciphertext message
c_j	The j-th bit of the ciphertext message
ctx_i	64-bit Message context string
$H(bl'_{Final})$	Recomputed final block

V. Lightweight Crypto-Based Encryption

This section introduces our proposed lightweight authenticated encryption scheme. The scheme is a modified version of the TinyJAMBU-128 [32] authenticated encryption technique. Data transmitted over the CAN bus is encrypted and authenticated using a small variant of JAMBU. It is implemented with 128-bit keyed permutation states and 64-bit associated data blocks. The following section provides a detailed description of the modified TinyJAMBU-128 authenticated encryption scheme to support the detection of replay attacks.

A. TinyJAMBU-128 Encryption for Secure Data Communication and Message Authentication

The proposed authenticated encryption utilizes TinyJAMBU mode with keyed permutation for randomizing the internal state of TinyJAMBU during encryption/ decryption.

In the classical version of TinyJAMBU-128 [32], the scheme was implemented by taking 64-bit associated data, a 64-bit plaintext message, and a 96-bit nonce as input parameters. TinyJMABU-128 [32] is comprised of four stages. Stage 1 is referred to as the initialization stage, encryption key and nonce are processed during this stage. The associated data string is used during the second stage of TinyJMABU-128 [32], it is utilized to update the current state of TinyJAMBU-128 [32] during encryption. The third stage involves data encryption where a ciphertext message is computed. Authentication tag generation and verification steps were performed during the finalization stage.

In this research, we have modified the internal design of TinyJAMBU-128 [32] to include one additional stage, the associated data generation stage. The new stage was implemented and inserted between the initialization stage and the associated data processing stage which is referred to as stage 2 in the original TinyJAMBU-128 [32] algorithm. Besides integrating associated data processing into the TinyJAMBU-128, CAN message context computation was also incorporated into the modified TinyJAMBU-128. CAN message context computation was integrated into the TinyJAMBU-128 encryption/decryption engine. Each CAN data frame is bounded into a single message context prior to transmitting the frames over the CAN channel. Through the employment of 64-bits associated data strings within the TinyJAMBU-128 engine, a common message context can be easily computed by every ECUs on the network.

As illustrated in the classical version of TinyJAMBU-128 [32], associated data strings are supplied by the application for each data message encrypted and authenticated via TinyJAMBU-128. Therefore, to support the employment of our newly modified version of the TinyJAMBU-128 CAN bus network, we have added a new processing module that enables the establishment of associated data strings for each CAN frame that needs to be encrypted and authenticated. The data-associated module has no impact on the TinyJAMBU-128 encryption/ decryption and authentication stages. The newly modified version

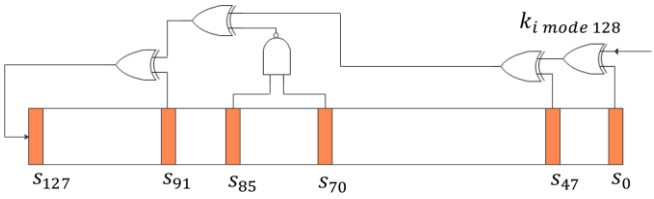

Fig. 4. Key permutation using a 128-bit nonlinear feedback shift reg.

of TinyJAMBU-128 should follow the same security analysis as the original version.

We have chosen TinyJAMBU-128 [32] due to its small state size and reduced encrypted message block size. The internal state of TinyJAMBU-128 was implemented with a 128-bit block compared to the 192-state size on JAMBU-128. TinyJAMBU-128 supports an encrypted message size of 32 bits which is half of the data length that a CAN frame can hold per transmission, two encrypted data messages can be transmitted over the bus via a single CAN data frame. Although, TinyJAMBU-128 supports three modes of operations with possible key sizes: 128-bit, 192-bit, and 256-bit, in this article, we have considered TinyJAMBU with a keyed-permutation size of 128-bit. The permutation of the state is based on the deployment of a 128-bit nonlinear feedback shift register. The following modules for TinyJAMBU were implemented and deployed on multiple STM32F411 [47] development boards:

- Keyed Permutation Module
- Initialization Module
- Associated Data Generation Module
- Associated Data Processing Module
- Data Encryption Module
- Authentication Tag processing Module
- Data Decryption Module
- Verification Module

1) Keyed Permutation Module: TinyJAMBU-128 relies on a 128-bit keyed permutation technique. The state of TinyJAMBU encryption is updated via the employment of a 128-bit nonlinear feedback shift register (see Fig. 4).

During each permutation round *i*, a combination of XOR, and NAND operations are performed on specific state's bits. The content of the state is then shifted by 1 bit to the left. In TinyJAMBU, *m* rounds are required to update the state. Algorithm 1 provides a detailed implementation of the keyed permutation module. The next section describes the initialization step of TinyJAMBU-128 [32].

2) Initialization Module: TinyJAMBU-128 [32] has been implemented on each ECU using a 128-bit key and 96-bit nonce. We have explored the utilization of multiple CAD IDs retrieved from previous CAN data frames to compute 64-bit associated data. To ensure the randomness of the bits within each computed associated data, bitwise operations that involve multiple associated data bits combined with a 64-bit nonlinear feedback shift register that shifts the data by one bit to left during each round were incorporated into TinyJAMBU-128 implementation. Initialization of TinyJAMBU-128 was implemented via bit randomization of the 128-bit state vector. By applying 1024

Algorithm 1: Keyed Permutation Module.

Input: 128-bit key: K ← $[k_0, k_1, \ldots, k_{127}]$
128-bit state: S ← $[s_0, s_1, \ldots, s_{127}]$
Permutation rounds: i

Output: Content of the state after being updated

TinyJAMBUStateUpdate (S, K, i):

1: $Temp \leftarrow s_0 \oplus s_{47} \oplus (\sim (s_{70} \wedge s_{85})) \oplus s_{91} \oplus k_{i\ mod\ 128}$
2: **for** j in range (*StateLength*-1) **do:** //*StateLength* = 127
3: $s_j \leftarrow s_{j+1}$ //shift the content of S by 1-bit to the left
4: **end for**
5: $s_{127} \leftarrow Temp$
6: **end of TinyJAMBUUpdate**

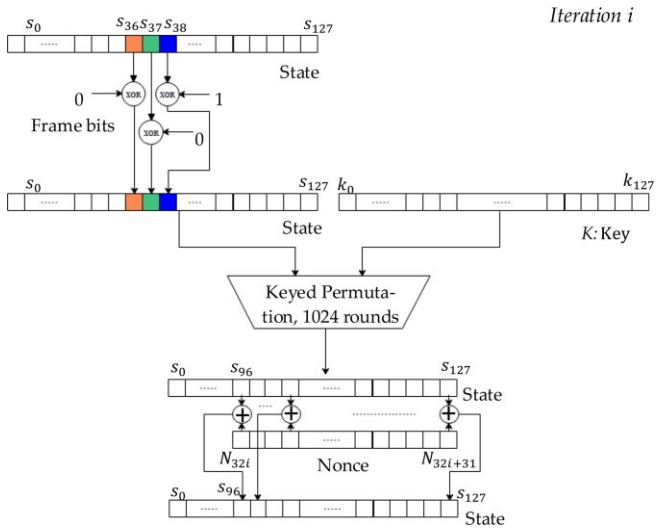

Fig. 5. i-th iteration of TinyJAMBU's Initialization step.

keyed permutation rounds on the state, a highly randomized 128-bit state is computed. During each round, bits [00,1] are xored with the state's bits s_{36}, s_{37}, s_{38} respectively. State's bits are randomized by executing 640 keyed permutation rounds on the state. Finally, bits $\{s_{96}, s_{97}, \ldots, s_{127}\}$of the state are xored repeatedly with a 96-bit nonce. Fig. 5 provides a details implementation of the initialization modules.

3) Associated Data Generation Module: In the newly added stage, a 64-bit associated data string is computed for each CAN data frame sent over the bus. To link a current CAN data frame to its previously transmitted CAN messages, CAN ids for the five most recently transmitted CAN data frames are utilized to compute a 64-bit associated data string. The computed associated data string is used during TinyJAMBU-128 state update to support data encryption and message authentication for the currently transmitted CAN data frame. The following algorithm depicts a detailed implementation of the associated data generation module.

4) Associated Data Processing Module: Associated data *Ad* in TinyJAMBU is used to update the content of the state. Each data message transmitted over the CAN bus is linked to a

Algorithm 2: Associated Data Generation Module.

Inputs: $ID_{i\text{-}1}$ [11]← $ID_{i\text{-}1}$ [$id_0, id_1, \ldots, id_{10}$]
//CAN ids for the five most recently transmitted *frames*
$ID_{i\text{-}2}$ [11] ← $ID_{i\text{-}2}$ [$id_0, id_1, \ldots, id_{10}$]
$ID_{i\text{-}3}$ [11] ← $ID_{i\text{-}3}$ [$id_0, id_1, \ldots, id_{10}$]
$ID_{i\text{-}4}$ [11] ← $ID_{i\text{-}4}$ [$id_0, id_1, \ldots, id_{10}$]
$ID_{i\text{-}5}$ [11] ← $ID_{i\text{-}5}$ [$id_0, id_1, \ldots, id_{10}$]
Adtemp [64] ← $Adtemp[adtemp_0, .., adtemp_{63}]$
// Temporary array to hold associated data
$Adtemp[adtemp_0, adtemp_1, \ldots\ldots, adtemp_{63}]$ ← $Zero[00, \ldots\ldots, 0]$ //*Adtemp string is initialized to zero*

Output: Ad_i [64] //Associated data string for i-th frame

AssociatedDataGeneration (Adtemp [64], $ID_{i\text{-}1}$ [11], $ID_{i\text{-}2}$ [11], $ID_{i\text{-}3}$ [11], $ID_{i\text{-}4}$ [11], $ID_{i\text{-}5}$ [11]):

1: $Adtemp\ [adtemp_0, adtemp_1, \ldots\ldots, adtemp_{10}]$ ← $ID_{i-1}[id_0, id_1, \ldots\ldots, id_{10}]$
2: $Adtemp\ [adtemp_{11}, adtemp_1, \ldots\ldots, adtemp_{21}]$ ← $ID_{i-2}[id_0, id_1, \ldots\ldots, id_{10}]$
3: $Adtemp\ [adtemp_{22}, adtemp_1, \ldots\ldots, adtemp_{32}]$ ← $ID_{i-3}[id_0, id_1, \ldots\ldots, id_{10}]$
4: $Adtemp\ [adtemp_{33}, adtemp_1, \ldots\ldots, adtemp_{43}]$ ← $ID_{i-4}[id_0, id_1, \ldots\ldots, id_{10}]$
5: $Adtemp\ [adtemp_{44}, adtemp_1, \ldots\ldots, adtemp_{54}]$ ← $ID_{i-5}[id_0, id_1, \ldots\ldots, id_{10}]$
6: **for** i ← 0 to 9 **do:** // *nonlinear shift register*
7: $Adtemp[adtemp_{55}] \leftarrow Adtemp[adtemp_7] \oplus Adtemp[adtemp_{54}] \oplus Adtemp[adtemp_{37}]$
8: $Adtemp[adtemp_{56}] \leftarrow Adtemp[adtemp_1] \oplus Adtemp[adtemp_{10}] \oplus Adtemp[adtemp_{36}]$
9: $Adtemp[adtemp_{57}] \leftarrow Adtemp[adtemp_2] \oplus Adtemp[adtemp_{17}] \oplus Adtemp[adtemp_{34}]$
10: $Adtemp[adtemp_{58}] \leftarrow Adtemp[adtemp_{15}] \oplus Adtemp[adtemp_{23}] \oplus Adtemp[adtemp_{13}]$
11: $Adtemp[adtemp_{59}] \leftarrow Adtemp[adtemp_{40}] \oplus Adtemp[adtemp_{27}] \oplus Adtemp[adtemp_{25}]$
12: $Adtemp[adtemp_{60}] \leftarrow Adtemp[adtemp_9] \oplus Adtemp[adtemp_{31}] \oplus Adtemp[adtemp_{20}]$
13: $Adtemp[adtemp_{61}] \leftarrow Adtemp[adtemp_{21}] \oplus Adtemp[adtemp_{39}] \oplus Adtemp[adtemp_{44}]$
14: $Adtemp[adtemp_{62}] \leftarrow Adtemp[adtemp_{49}] \oplus Adtemp[adtemp_{29}] \oplus Adtemp[adtemp_5]$
15: $Adtemp[adtemp_{63}] \leftarrow Adtemp[adtemp_{15}] \oplus Adtemp[adtemp_{11}] \oplus Adtemp[adtemp_{19}]$
16: **for j** ← 0 to 63 **do:**
17: Ad_i [j+1] ← $Adtemp[j]$
18: **end for:**
19: Ad_i [0] ← $Adtemp$[63]
20: **for j** ← 0 to 63 **do:**
21: $Adtemp[j] \leftarrow Ad_i$ [j]
22: **end for:**
23: **end for:**
24: **end of AssociatedDataGeneration**

randomly computed 64-bit associated data. Existing implementation of TinyJAMBU-128 enables the processing of 64-bit associated data strings and plaintext data as input parameters to the encryption/decryption algorithms. In this research, associated data strings are constructed by concatenating multiple 11-bit

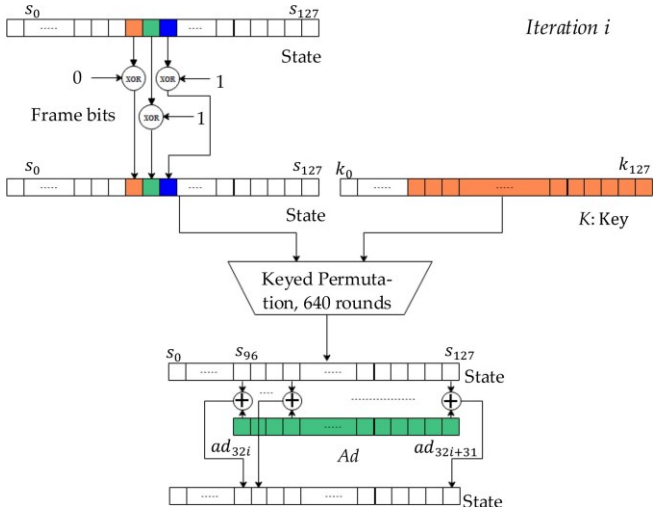

Fig. 6. i-th iteration of TinyJAMBU's associated data processing.

CAN IDs. A total of 55 bits were extracted from CAN IDs fields from the five most recent CAN data frames. Bits are randomized via the application of several bitwise xor operations with bit permutation and bitwise rotation. Since we randomize the bits of each computed 64-bit associated data string, every CAN frame should have a different associated data string. A 64-bit nonlinear feedback shift register is implemented to support the establishment of associated data strings. After the initialization step, the state's bits s_{36}, s_{37}, and s_{38} are bitwise xored with bits [0,1], and 1 respectively. Contents of the state are then bit permutated by employing 640 rounds of keyed permutation function. After permutating the state's bits, 32-bit of the associated data string is xored with the state. The processing of associated data is presented in Fig. 6.

5) Data Encryption Module: CAN data of length 64-bit is encrypted via the TinyJAMBU data encryption algorithm. After randomizing the content of the state during the associated data processing module and initialization module, 64-bit CAN data is passed to the data encryption module. During the data encryption step, the state's bits s_{36},s_{37}, s_{38} are xored with bits [1,0,1] respectively. State bits are manipulated using 640 rounds of the keyed permutation function. State's bits $\{s_{96}, \ldots, s_{127}\}$are updated by xoring the current state bits $\{s_{96}, \ldots, s_{127}\}$with CAN message plaintext's bits $\{m_{32i}, \ldots, m_{32i+31}\}$. The final ciphertext's bits are computed by xoring state's bits $\{s_{64}, \ldots, s_{95}\}$ with the plaintext's bits $\{m_{32i}, \ldots, m_{32i+31}\}$.

6) Authentication Tag Processing Module: Authenticated encryption is supported by TinyJAMBU-128 [32] by generating a 64-bit authentication tag τ_{M_i} for CAN message M_i.The authentication tag is transmitted with the encrypted CAN data frame. In the proposed system, computed authentication tags are transmitted over the CAN bus using a single CAN frame with a predefined ID (2047). For ECUs to be able to recognize that a CAN frame contains an authentication tag and not a payload, a special CAN ID is assigned. If an ECU received a CAN frame with ID 2047, the ECU will be able to recognize and process it accordingly as an authentication tag. Authentication tags were

Algorithm 3: Data Encryption Module.

Input: *keylen* ← 128 //*length of the key*
Mlen gth ← 64 //*length of CAN message*
K [*keylen*]← [$k_0, k_1, \ldots, k_{127}$]
S [*keylen*] ← [$s_0, s_1, \ldots, s_{127}$] //*updated during the associated data processing step*
M [*mlength*]←[$m_0, m_1, \ldots, m_{63}$]
Output: C [$c_0, \ldots, c_{63}$] //*ciphertext*
Data Encryption (S, K, M):
1: **for** $j \leftarrow 0$ to $\lfloor \frac{mlen}{32} \oplus$ **do**:
2: $s_{36} \leftarrow s_{36} \oplus 1$
3: $s_{37} \leftarrow s_{37} \oplus 0$
4: $s_{38} \leftarrow s_{38} \oplus 1$
5: *TinyJAMBUStateUpdate (S, K, 1024)*
6: S [$s_{96}, s_{97}, \ldots, s_{127}$]$\leftarrow S[s_{96}, s_{97}, \ldots, s_{127}] \oplus m[m_{32j}, \ldots, m_{32j+31}]$
7: C [$c_{32i}, c_{32i+1}, \ldots, c_{32i+31}$]$\leftarrow S[s_{64}, s_{65}, \ldots, s_{95}] \oplus m[m_{32j}, \ldots, m_{32j+31}]$
8: return C
9: **end for**
10: **end of Data encryption**

Algorithm 4: Authentication Tag Processing Module.

Input: K [128]← [$k_0, k_1, \ldots, k_{127}$]
S [*keylen*] ← [$s_0, s_1, \ldots, s_{127}$] //*updated during the encryption step*
Output: τ_{M_i} [64]←[$\tau_0, \tau_1, \ldots, \tau_{63}$]
Authentication tag processing (S, K):
1: $s_{36} \leftarrow s_{36} \oplus 1$
2: $s_{37} \leftarrow s_{37} \oplus 1$
3: $s_{38} \leftarrow s_{38} \oplus 1$
4: *TinyJAMBUStateUpdate (S, K, 1024)*
5: $\tau_{M_i}[\tau_0, \tau_1, \ldots, \tau_{31}] \leftarrow S[s_{64}, s_{65}, \ldots, s_{95}]$
6: $s_{36} \leftarrow s_{36} \oplus 1$
7: $s_{37} \leftarrow s_{37} \oplus 1$
8: $s_{38} \leftarrow s_{38} \oplus 1$
9: *TinyJAMBUStateUpdate (S, K, 640)*
10: $\tau_{M_i}[\tau_{32}, \tau_{33}, \ldots, \tau_{63}] \leftarrow S[s_{64}, s_{65}, \ldots, s_{95}]$
11: return τ_{M_i}
12: **end of Authentication tag processing**

not encrypted. They were transmitted over CAN bus in plaintext messages.

Authentication codes are computed based on applying multiple keyed permutations on the state. The first 32 bits of the authentication code are generated by updating the state's bits using 1024-keyed permutation rounds. The rest of the authentication code bits $\{\tau_{32}, \ldots, \tau_{63}\}$ is established through the employment of 640 keyed permutation rounds on the state's bits. Algorithm 3 introduces the Authentication tag processing.

Algorithm 5: Data Decryption Module.

Input: *keylen* ← 128 //*length of the key*
Clength ← 64 //*length of the ciphertext*
K [*keylen*]← [$k_0, k_1, \ldots, k_{127}$]
S [*keylen*] ← [$s_0, s_1, \ldots, s_{127}$] //*updated during the associated data processing step*
C [Clength]←[$c_0, c_1, \ldots, c_{63}$]
Output: M [$m_0, \ldots, m_{63}$] // *plaintext CAN message*
Data Decryption (S, K, C):
1: **for** j ← 0 to $\lfloor \frac{mlen}{32} \oplus$ **do**:
2: $s_{36} \leftarrow s_{36} \oplus 1$
3: $s_{37} \leftarrow s_{37} \oplus 0$
4: $s_{38} \leftarrow s_{38} \oplus 1$
5: *TinyJAMBUStateUpdate (S, K, 1024)*
6: M [$m_{32i}, m_{32i+1}, \ldots, m_{32i+31}$]$\leftarrow S[s_{64}, s_{65}, \ldots, s_{95}] \oplus C[c_{32j}, \ldots, c_{32j+31}]$
7: S [$s_{96}, s_{97}, \ldots, s_{127}$]$\leftarrow S[s_{96}, s_{97}, \ldots, s_{127}] \oplus M[m_{32j}, \ldots, m_{32j+31}]$
8: return M
9: **end for**:
10: **end of Data Decryption**

7) Data Decryption Module: During the data decryption step, initialization and associated data processing modules are instantiated to update the state of TinyJAMBU-128 [32]. Similar to data encryption, a 3-bit value of 101 is xored with the state's bits s_{36}, s_{37}, and s_{38} and the contents of these bits are updated accordingly to the results of the xor operation. Then, 1024 rounds of the keyed permutation step are applied to the state. During the first iteration of the data decryption module, bits $\{m_0, \ldots, m_{31}\}$ are computed by xoring ciphertext bits $\{c_0, \ldots, c_{31}\}$with state's bits $\{s_{64}, \ldots, s_{95}\}$. The state's bits $\{s_{96}, \ldots, s_{127}\}$ are updated by xoring the current state's bits contents $\{s_{96}, \ldots, s_{127}\}$ with the computed plaintext bits $\{m_0, \ldots, m_{31}\}$. Similarly, plaintext bits $\{m_{32}, \ldots, m_{63}\}$ is computed by xoring these bits with the state's bits $s_{64}, \ldots, s_{95}$ and then use the generated plaintext bits to modify the current content of the state's bits $\{s_{96}, \ldots, s_{127}\}$. The following provides a detailed implementation of the decryption module.

8) The Verification Module: During the verification step, a CAN data message M_i can be authenticated by computing a 64-bit authentication tag $\tau^t_{M_i}$ and compare it with the received authentication tag τ_{M_i}. If $\tau^t_{M_i} = \tau_{M_i}$ the message is authenticated and accepted, otherwise, the received message is rejected. Algorithm 7 provides a full implementation for the verification module. In the case of miss verification, an error counter is incremented for each miss verification. We also keep a record of the CAN frame ID. If the error counter exceeds a predefined threshold value, the system will send an alert message to all ECUs connected to the CAN bus. CAN messages that are not verified will be ignored.

Algorithm 6: Verification Module.

Input: *Taglength* ← 64 *//length of the authentication tag*
K [128]← [$k_0, k_1, \ldots, k_{127}$]
S [*keylen*] ← [$s_0, s_1, \ldots\ldots, s_{127}$]
// updated during the initialization step τ_{M_i}
[Taglength]←[$\tau_0, \tau_1, \ldots, \tau_{63}$]

Output: return true if the received tag matches the computed tag, otherwise return false.

Verification (S, K, τ_{M_i}):

1: $s_{36} \leftarrow s_{36} \oplus 1$
2: $s_{37} \leftarrow s_{37} \oplus 1$
3: $s_{38} \leftarrow s_{38} \oplus 1$
4: *TinyJAMBUStateUpdate (S, K,* 1024)
5: $\tau_{M_i}[\tau_0, \tau_1, \ldots, \tau_{31}] \leftarrow S[s_{64}, s_{65}, \ldots, s_{95}]$
6: $s_{36} \leftarrow s_{36} \oplus 1$
7: $s_{37} \leftarrow s_{37} \oplus 1$
8: $s_{38} \leftarrow s_{38} \oplus 1$
9: *TinyJAMBUStateUpdate (S, K,* 640)
10: $\tau_{M_i}[\tau_{32}, \tau_1, \ldots, \tau_{63}] \leftarrow S[s_{64}, s_{65}, \ldots, s_{95}]$
11: if $\tau_{M_i} == \tau^{t}_{M_i}$ return true
12: **end of Verification**

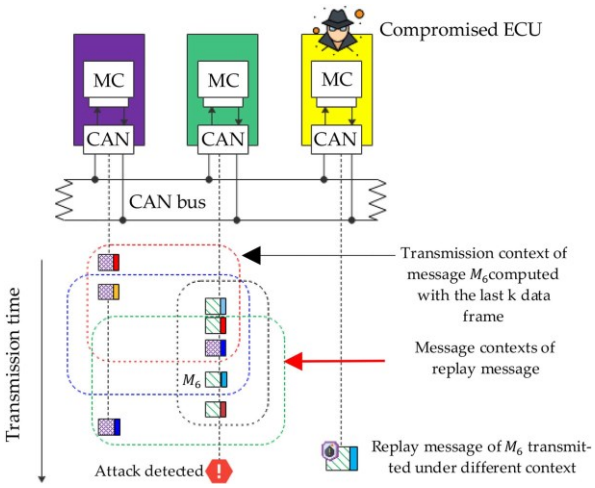

Fig. 7. CAN message replay threat modeling.

VI. CAN Replay Attack and The Proposed Countermeasure Mechanism

In this research, we have considered the development of a countermeasure mechanism against replay attacks on CAN. The proposed mechanism enables ECUs of detecting the presence of replay threats occurring on CAN. The following provides full descriptions of CAN replay threat modeling and its countermeasurement approach.

A. CAN Message Replay Threat Modeling

We have considered the implementation of simulated CAN message replay attacks through the deployment of a malicious ECU that is capable of capturing every CAN data frame sent over the bus and replaying them at different times (see Fig. 7). We have applied a random process to inject replay attack messages into the CAN bus network at various injection rates (2% - 50%). Simulated CAN message replay attacks were tested on a real testbed that is comprised of three ECUs attached to CAN. Two of the three nodes were considered benign, while the third unit was acting as an adversary. Replay attacks were conducted on the proposed testbed with transmitted CAN data frames being encrypted and authenticated via TinyJAMBU-128. During simulation, malicious node continuously sniffs CAN messages from the bus and retransmits these messages during a different time interval. Since authenticated encryption provides zero resiliency against replay threats, we have developed a technique that extends the security features of TinyJAMBU-128 to support dynamic detection capability against CAN message replay attacks.

B. Countermeasure Mechanism Against Replay Threats

Dynamic detection of CAN message replay attacks was implemented via the integration of CAN message context awareness capability. Based on the type of event, different CAN data frames are computed and disseminated over the network. In the proposed approach, we assumed that each transmitted CAN message is associated with a single context. A message context is established by using a sequence of *k* historical CAN data frames. By observing the state of the CAN channel during the last *k* active transmissions, communicating ECUs will be capable of constructing a common 64-bit binary string that could be used to represent the context of the next transmitted CAN data frame. In the proposed approach, a message context is constructed based on the CAN IDs of the last *k*-data frames transmitted over the bus. To send the next CAN data frame, a message context is computed first by the sender and encoded into a 64-bit associated data string. Which is hence used to encrypt/decrypt CAN data frames. Similarly, the message context presented by the 64-bit associated data string is computed on the receiving node and used during message authentication and data decryption. Our proposed technique enables the establishment of secured bonds between every CAN data frame and the contexts in which these data frames were transmitted. Since every encrypted CAN data frame is securely bounded into its message context, encrypted CAN frames transmitted during different contexts will be detected as CAN message replay threats. The following provides a high-level implementation of how a message context is computed:

C. Establishment of Message Context Between Two ECUs

CAN message context computation was integrated into the TinyJAMBU-128 encryption/decryption engine. Each CAN data frame is bounded into a single message context prior to transmitting the frames over the CAN channel. When a CAN frame is transmitted over the CAN bus, a CAN message context is constructed. The CAN frame's message context is established by using a sequence of *k* historical CAN data frames. By observing the state of the CAN channel during the last *k* active transmissions, all communicating ECUs will be capable of constructing a common 64-bit binary string that could be used

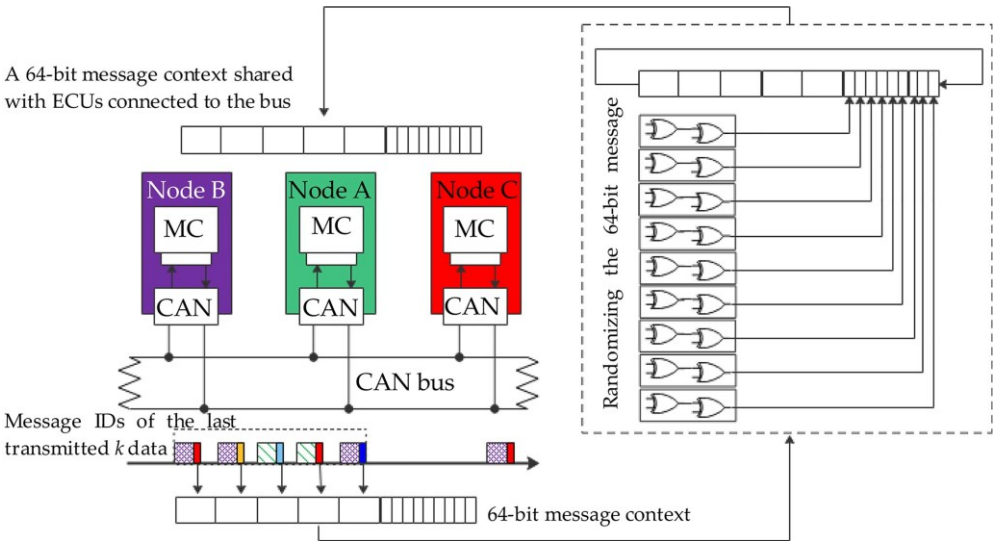

Fig. 8. Message context computation.

to represent the context of the next transmitted CAN data frame. For ECU nodes to be able to establish the first CAN message context, initially, all ECUs are preloaded with the same orphan block.

Through the employment of 64-bits associated data strings within the TinyJAMBU-128 engine, a common message context can be easily computed by every ECUs on the network. A receiver node can leverage message context information to determine whether the received CAN data frame is generated due to a replay attack or due to an actual event. For example, two ECUs nodes, node A and node B, where node A is acting as a sender and node B as a receiver. Prior to any data transmission, both sender and receiver nodes are required to compute a common 64-bit message context string ctx_i. The computed context message will be encoded into an associated data string that will be utilized by the TinyJAMBU-128 crypto engine to encrypt/decrypt the CAN data frame M_i. The following protocol's steps are required during a CAN message context computation process (see Fig. 8).

1) CAN messages IDs of the last k transmitted data frames $\{ID_{i-1}, \ldots, ID_{i-5}\}$ are recorded by all ECUs including nodes A & B.
2) Both sender and receiver copy the recorded CAN messages IDs into bit 0 through bit 54 of the message context string ctx_i.
3) Bit 55 through bit 63 of the message context ctx_i will be computed by applying multiple bitwise xor operations on the content of ctx_i.
4) A nonlinear shift register is employed on the content of ctx_i to randomize its bits.
5) CAN message context ctx_i is encoded into a 64-bit associated data string by the sender and fed into the TinyJAMBU-128 crypto engine.
6) The sender encrypts the CAN data frame by applying TinyJAMBU-128 with the computed message context and transmits the encrypted data frame over the CAN channel.
7) TinyJAMBU-128 will be executed on the receiver with the computed associated data string to decipher the received message.

VII. Data Traceability Via Block Chaining and Hash Computation

Data traceability capability was supported by the proposed scheme via the employment of a block-chaining approach. Encrypted and authenticated CAN data frames received by ECU nodes are encoded into data blocks. Since the CAN bus is a broadcast medium, for every CAN data frame sent over the bus, each ECU node computes its copy of the data block. Data blocks are chained together to form a common blockchain. Copies of the computed blockchain are stored at every ECU node attached to the CAN bus. The final block in the chain serves as an integrity check for all previous data blocks. Data tracing and data integrity were supported via blockchain data validation. Blockchain data stored in ECU nodes can be utilized by the system control unit to identify (i) abnormal behaviors occurring during system operations. For example, a malfunctioning ECU node injects faulty data into the system bus. (ii) anomalous CAN data frames injected by a malicious ECU node. The proposed block-chaining protocol is based on a two-step process (see Fig. 9).

A. Data Blocks Construction Scheme

During system operation, when a CAN data frame D_i with id, ID_i received by ECU nodes $\{ECU_1, ECU_2, \ldots, ECU_n\}$. Every ECU node processes the received data frame as follows:

- Each node computes a hash value on the CAN data frame D_i, $H(D_i \parallel H(bl_{i-1}) \parallel TID_i \parallel T_i)$, where $H(bl_{i-1})$ represents the hash of the previous block, TID_i is the transaction id for block i, and T_i is data transmission timestamp. If i = 0, the hash of the previous block $H(bl_0)$ will be set to the initialization vector which is called the orphan block.

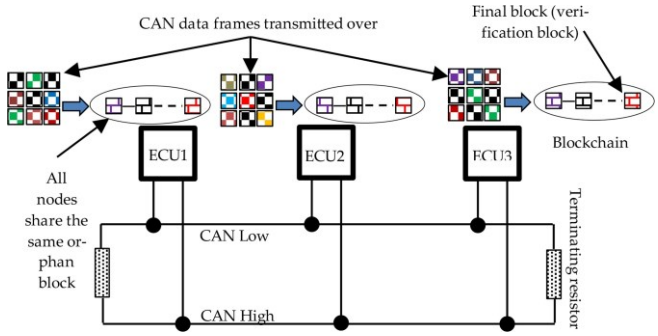

Fig. 9. Block chaining in CAN bus for supporting data integrity and secure data traceability.

- CAN data frame D_i along with the computed data block bl_i is saved in every ECU node.
- The final block (leaf block) will be used to provide secure data traceability validation and data integrity checks for every data frame transmitted over the bus.

B. Blockchain Validation Scheme

CAN data frames transmitted over the bus are captured and encoded into a single blockchain. The computed blockchain is shared and stored at every ECU node communicated over the bus. Our proposed system offers full resiliency against data modification and data injection attacks via block validation of the final block in the chain. Due to the blockchain computing characteristic, a successful data injection attack requires full modification of the current data block including all subsequent data blocks, which makes such attacks infeasible. Finally, the final block in the chain can be used to verify the integrity of every CAN data frame transmitted over the bus. In the case of a malfunctioning CAN-enable system, blockchain data collected from different ECUs can be used to recompute the final block $H(bl'_{Final})$. The recomputed final block $H(bl'_{Final})$ is then compared with the content of the stored final block $H(bl_{Final})$. If the two final blocks match, then the integrity of the data trace is verified, and the collected CAN data can be used for further analysis. Our proposed system offers full resiliency against data modification and data injection attacks via block validation of the final block in the chain. Due to the blockchain computing characteristic, a successful data injection attack requires full modification of the current data block including all subsequent data blocks, which makes such attacks infeasible. Finally, the final block in the chain can be used to verify the integrity of every CAN data frame transmitted over the bus. In the case of a malfunctioning CAN-enable system, blockchain data collected from different ECUs can be used to recompute the final block $H(bl'_{Final})$. The recomputed final block $H(bl'_{Final})$ is then compared with the content of the stored final block $H(bl_{Final})$. If the two final blocks match, then the integrity of the data trace is verified, and the collected CAN data can be used for further analysis.

Algorithm 7: BlockChaining Verification.

Inputs: $i \leftarrow 0$ //Current data block index. Initially thisvalue is set to zero to represent an orphan block bl_0 D_i //Current CAN data fame transmitted over the bus and received by all ECUs
T_i // Timestamp of current CAN data frame D_i
ID_i // Id of the current CAN data frame D_i
$H'(bl_i)$ // block i computed by the sender ECU and transmitted along CAN data frame D_i and its timestamp data T_i
$H(bl_0)$ // Hash value computed on the orphan block
$bl_0.H(bl_0)$. is preloaded into every ECU connected to the CAN bus

Outputs: return true for successful verification and false for missed verification
BlockChainingVerification (i, D_i, T_i, ID_i, $H'(bl_i)$:

1: **if** ($i == 0$)**:**
// $ECU_{j\leftarrow 0}$, $ECU_{j\leftarrow 1}$,..., $ECU_{j\leftarrow n}$ recompute their first blocks
in the chain by applying the preloaded orphan block $H(bl_0)$. Computed block values are stored internally.
2: $H(bl_1)_{ECU_{j\leftarrow 0}} \leftarrow H(D_0||H(bl_0)||ID_0||T_0)$
$H(bl_1)_{ECU_{j\leftarrow 1}} \leftarrow H(D_0||H(bl_0)||ID_0||T_0)$
.
.
$H(bl_1)_{ECU_{j\leftarrow n}} \leftarrow H(D_0||H(bl_0)||ID_0||T_0)$
// Every $ECU_{j\leftarrow x}$ will compare it computed block $H(bl_1)_{ECU_{j\leftarrow x}}$ with the received block $H'(bl_0)$
3: **If** ($H(bl_1)_{ECU_{j\leftarrow x}} \neq H'(bl_1)$) **return** false
// If one ECU invalidates the received block, it broadcast an alert message over the CAN bus. ECUs that received the first alert message will suspend its processing for the current block.
4: **end if:**
5: **else:**
// $ECU_{j\leftarrow 0}$, $ECU_{j\leftarrow 1}$,..., $ECU_{j\leftarrow n}$ recompute the current block in the chain by applying the previous $H(bl_{i-1})$ and store the computed values in their internal memory.
6: $H(bl_i)_{ECU_{j\leftarrow 0}} \leftarrow H(D_i||H(bl_{i-1})||ID_i||T_i)$,
$H(bl_i)_{ECU_{j-1}} \leftarrow H(D_i||H(bl_{i-1})||ID_i||T_i)$,
.
.
$H(bl_i)_{ECU_{j\leftarrow n}} \leftarrow H(D_i||H(bl_{i-1})||ID_i||T_i)$
// Every $ECU_{j\leftarrow x}$ will compare it computed block $H(bl_i)_{ECU_{j\leftarrow x}}$ with the received block $H'(bl_i)$
7: **If** ($H(bl_i)_{ECU_{j\leftarrow x}} \neq H'(bl_i)$) **return** false //If one ECU validates a received block, it broadcast an alert message over the CAN bus. ECU received the first alert message and will suspend processing the current block.
8: **end else**
9: **return true**
10: **end of the BlockChainingVerification**

VIII. Security Analysis

Inspired by the works of Ding Wang [41], Qingxuan [42], and Neal Koblitz [43], we have followed a similar approach in analyzing the security properties of the modified TinyJAMBU-128 against CAN message replay attacks. However, cryptoanalysis based on "provable security" has been exploited in the original work of TinyJAMBU-128 [32], it is limited to two adversary models (i) nonce-respecting attacks on TinyJAMBU-128 privacy (ii) nonce-reuse attacks on TinyJamu-128 authenticity. As suggested in the works of Ding Wang and Qingxuan [41], [42], [43], security protocols that are provably secure under some cryptographic assumptions imposed by the designer usually fail to capture all the different aspects of an adversary model. For example, CAN replay attacks cannot be captured by exiting adversarial models presented in TinyJAMBU-128. Also, as we change the usage dynamic of TinyJAMBU-128 from securing data in a constrained environment to authenticating and encrypting messages over CAN bus network, a new systematical security assessment methodology with various system performance metrics has been developed in this work. Our threat analysis was not based on a formal probabilistic attack modeling technique, rather it involves the deployment of real adversarial ECU that inject CAN replay messages over the bus with varying injection rates. Using a real CAN-enable testbed, we have analyzed the modified TinyJAMBU-128 based on data privacy, authenticity, and resiliency against CAN replay attacks.

IX. Performance Analysis

CAN data frames are usually transmitted over the bus at the rate of 10 msec to 500 msec. Based on the underlying system, processing times for CAN frames are highly dependent on the underlying processing power of each ECU. ECUs that handle time-critical tasks are often integrated with a high-end processer capable of processing messages at the rate of 0.1 msec. Meanwhile, ECUs that handle non-time-sensitive data usually have a data processing time of 100 msec to 500 msec. In our system simulation, we have considered the employment of ECUs with low-end processor power capability. The proposed security system offers a tradeoff between reliable data security with traceability and message delivery time. This section presents performance analysis of TinyJAMBU-128 lightweight authenticated encryption, AES-128, AES-192, AES-256 [44], ARIA-128, ARIA-192, ARIA-256 [45], Camellia-128, Camellia-192, and Camellia-256 [46]. For AES [44], ARIA [45], and Camellia [46], message authentication was incorporated into these crypto engines via the full employment of SHA-512.

To compare the performance of modified TinyJAMBU-128 against similar symmetric block cipher encryption schemes, processing time and message overhead for AES, ARIA, and CAMELLIA data encryption schemes have been captured and analyzed during this effort. Since modified TinyJAMBU-128 supports authenticated encryption inherently, data authentication capability for CAN messages encrypted by AES, ARIA, or CAMELLIA schemes has been achieved via the employment of SHA-512. AES is based on substitution-permutation network, it supports different numbers of rounds. AES-128 uses 10 rounds, AES-192 involves 12 operational rounds, meanwhile, data encrypted with AES-256 requires 14 rounds. During each round, different operations are involved which include, byte substitution, shift-rows permutation, mixcolumns, and addroundkey. ARIA uses similar technique for enciphering data, it uses a substitution-network based on AES [44]. ARIA [45] is capable of supporting three modes of encryptions with different key sizes, 128, 192, and 256 bits. Depending on the key size, data encrypted/decrypted by ARIA requires 12 rounds, 14 rounds, or 16 rounds. ARIA's main encryption/decryption engine is comprised of two submodules (i) key scheduling and (ii) data randomizing. CAMELLIA [46] is another Feistel cipher, the algorithm performs 18 rounds when a 128-bit key is used for encryption data. Meanwhile, data encrypted using CAMELLIA-192 or CAMELLIA-256 only require 24 rounds. The main design of CAMELLIA includes the "F-function" and the "FL-function". The F-function takes 128-bit inputs and mixes them with the round key. During the F-function call, a single block is computed. For every six-round block, the algorithm calls the FL-function where a logical transformation is applied.

For all tested encryption/decryption algorithms, message authentication data were pushed into the CAN bus as separate CAN data frames. In the proposed testbed, data frames authenticated via SHA-512 were fragmented into 8 data frames. Since each CAN data frame can only support a data length of bytes per frame, each CAN data will require 8 data frames for authentication to enable authenticated encryption on AES, ARIA, and Camellia. Meanwhile, TinyJAMBU-128 supports data authentication via a single 64-bit authentication tag. A data frame encrypted via TinyJAMBU-128 requires only a single CAN data frame to achieve message authentication. Our simulation results were based on the observation of all CAN data frames including authentication tags transmitted over the network.

During this work, we investigated CAN bus load percentage, data processing time, and message overhead for each of the deployed crypto algorithms. In our experiment, CAN bus efficiency based on bus load and message overhead was estimated under various CAN data transmission rates and a 300KHz CAN frequency. CAN bus load measurements and message overhead were captured for TinyJAMBU-128, AES-128, AES-192, AES-256, ARIA-128, ARIA-192, ARIA-256, Camellia-128, Camellia-192, and Camellia-256 under continuous data transmission, 1msec, and 10msec CAN data frames transmission rates.

Fig. 10 shows CAN bus load percentages for TinyJAMBU-128, AES, ARIA, and Camellia under continuous data transmission. Based on our simulation results, the AES encryption variant achieved the highest bus load percentages compared to other data-authenticated encryption modules. The average CAN bus load measured across the three AES variants was approximately 880.415% as compared to ARIA, and Camellia variants which achieved average CAN busloads of 743.38% and 779.211% respectively. Meanwhile, the CAN bus load reached 0.42195% while running TinyJAMBU-128. Similarly, Figs. 11 and 12 illustrate the network performance in terms of bus load percentages computed at 1 msec and 10 msec data transmission rates respectively. CAN bus load measurements were collected

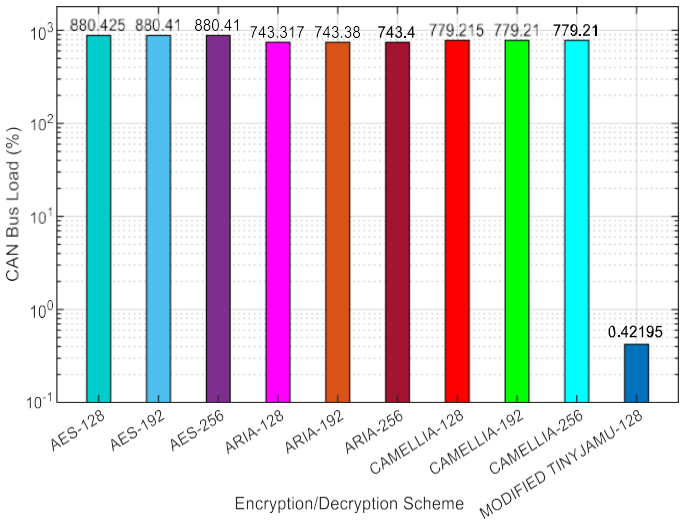

Fig. 10. Percentage of CAN bus load captured under various data encryption with continuous data transmission.

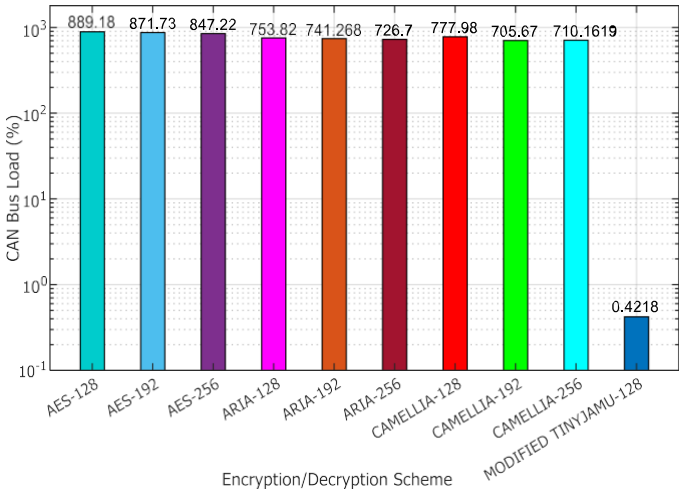

Fig. 11. Percentage of CAN bus load captured under various encryption algorithms with 1 data frame sent every 1 msec.

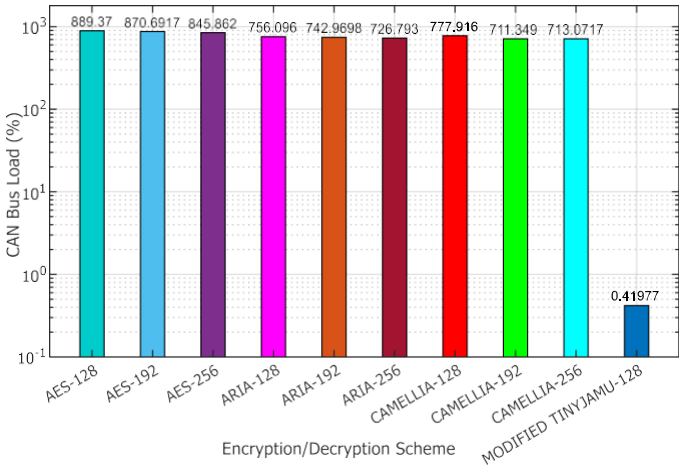

Fig. 12. Percentage of CAN bus load captured under various encryption schemes with 1 data frame sent every 10 msec.

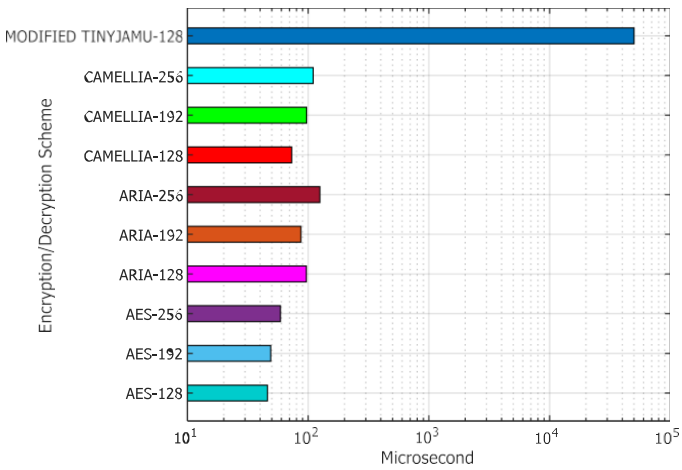

Fig. 13. Processing time for various encryption techniques tested on CAN-enabled devices.

while running the proposed modified TinyJAMBU-128 encryption. It shows that our CAN network testbed exhibited minimal bus load as compared to AES, ARIA, and Camellia. Higher CAN bus loads were recorded while running AES, ARIA, or Camellia data. Every CAN frame required 8 additional CAN frames for authentication. In the case of TinyJAMBU-128, each CAN frame requires only one additional authentication frame.

Data processing for each encryption scheme has been measured. Processing time was measured based on how much time is required to encrypt or decrypt a single CAN data frame on the hardware. As shown in Fig. 13, AES variants have the best processing time compared to other encryption schemes. Due to a large number of encryption rounds, TinyJAMBU-128 has the highest computation time. With a processing time of 50 msec per CAN data frame encryption, TinyJAMBU-128 remains a feasible encryption solution for most of today's CAN bus-based network systems.

Average CAN bus message overheads were computed during the execution of AES, ARIA, Camellia, and TinyJAMBU-128. A total of 500 randomly generated messages were transmitted over the CAN bus. Three data transmission rates were tested. Fig. 14 presents message overhead observed during continuous data transmission. It shows that AES, ARIA, and Camellia introduce an average CAN bus message overhead of approximately 13.67. As we lower the data transmission rate from continuous to 1 msec (see Figs. 15 & 16), the average message overhead reduces to approximately 12.3 for AES, ARIA, and Camellia. In the case of a 10 msec data transmission rate, message overheads reach 10.98 when running AES, ARIA, or Camellia on the CAN network. For the various data transmission rates, TinyJAMBU-128 introduces the lowest message overhead. We have observed 2.15 message overhead under continuous data transmission, 2.14 and 2.062 under 1msec and 10msec data transmission rates respectively.

Finally, we have analyzed the sensitivity of the proposed countermeasure scheme against replay attacks. True Positive Rate (TPR) and False Positive Rate (FPR) of the anti-replay attacks model has been computed and presented in Figs. 17 and 18. To simulate a replay attack, we considered the deployment

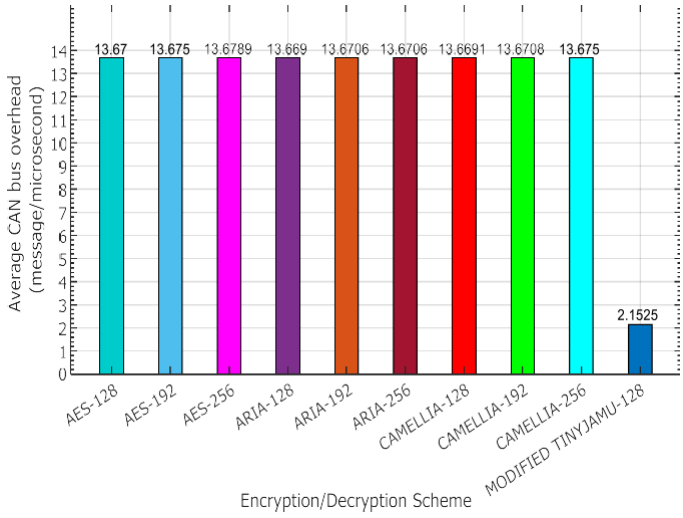

Fig. 14. Average CAN bus overhead captured under various encryption algorithms with continuous data transmission.

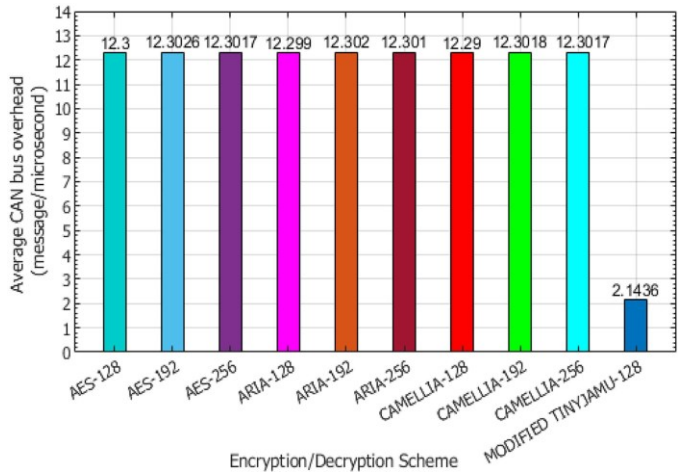

Fig. 15. Average CAN bus overhead captured under various encryption engines with 1 CAN frame sent every 1 msec.

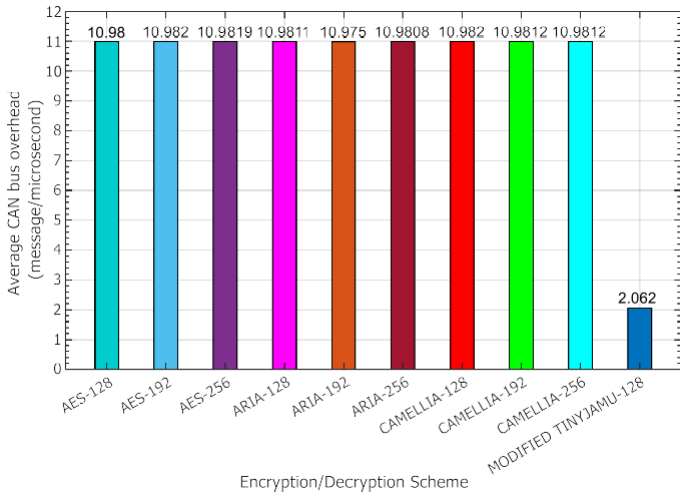

Fig. 16. Average CAN bus overhead captured under various encryption algorithms with a single CAN data frame transmitted every 10 msc.

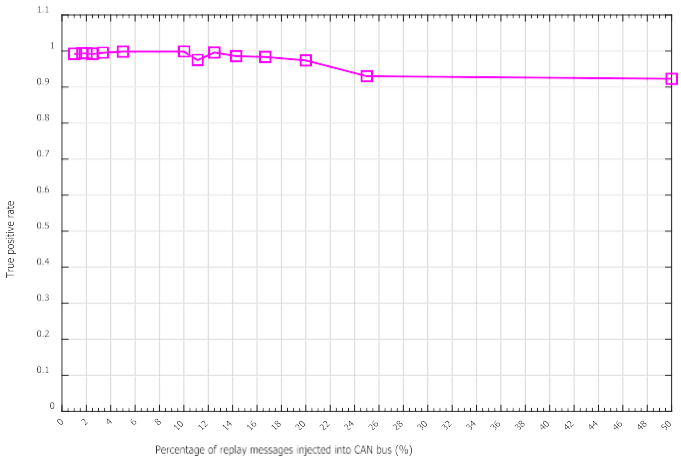

Fig. 17. CAN network's True Positive Rate (TPR) was collected under various percentages of replay attack messages.

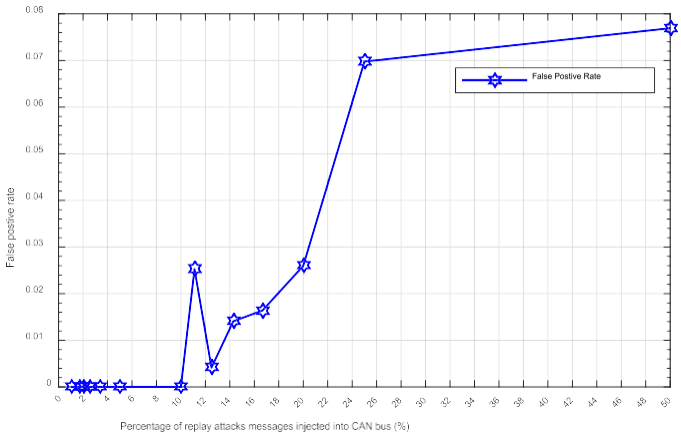

Fig. 18. CAN network's False Positive Rate (FPR) was collected under various percentages of replay attack messages.

of a malicious node. The attacking node sniffs CAN data frames from the bus and arbitrarily send replay messages over the bus. Fig. 17 shows the TPR of the proposed technique under various injection rates. With injection rates between 2% and 20%, our proposed scheme achieved TPR between 0.99 and 0.97. As we increase the injection rate from 20% to 50%, TPR decreases to 0.92. Fig. 18 shows the FPR of our anti-replay attack mechanism. Our performance results show that when the injection rate is lower than 10%, FPR was estimated as zero, as we increase the injection rate from 10% to 50%, FPR increases to 0.076. Based on the performance analysis of the proposed anti-replay attack model, the model established superiority in replay message detection.

X. The Proposed Testbed

To test the feasibility of the proposed CAN security subsystem, we proposed the development of a system that is comprised of three ECU development platforms with CAN capability. The three ECU nodes are based on the STM32 Nucleo board. The three ECU nodes are based on the STM32 Nucleo board. Each development board is equipped with the ARM 32-bit Cortex

M4 processor, an adaptive real-time accelerator, 80MHz maximum CPU frequency, 1MB flash programable memory, 128 KB SRAM, random number generator (TNG for HW entropy), multiple digital timers, three SPI, two I2C, two UART, 3 USART, and one CAN interface. We have employed the high-speed CAN transceiver MCP2551 to facilitate CAN data communication between the ECU nodes. MCP2551 supports a 1Mb/s transfer rate. It is implemented with ISO-11898 standard physical layer requirements. It supports a system with up to 112 nodes connected to the bus. Cryptographic algorithms that include AES, Camellia, Aria, modified TinyJAMBU-128, SHA512, and the replay attacks countermeasure mechanism were developed based on utilizing Mbed os API [48] and Mbed TLS library [49]. AES, Camellia, Aria, and SHA512 were supported by the Mbed TLS library. Meanwhile, the modified version of TinyJAMBU-128 and the countermeasure mechanism against replay attacks were fully implemented using Mbed os API. CAN data frames were created and pushed into the bus via CAN API.

XI. Conclusion

This article proposed the development and integration of a crypto-based engine that supports lightweight authenticated data encryption and secure data traceability capabilities for systems communicating over the CAN bus network. As opposed to classical encryption/decryption schemes that require heavy computation and high communication overheads for supporting authenticated encryption. Our proposed scheme is based on TinyJAMBU-128 authenticated data encryption. It maintains low memory usage and minimal communication overhead which makes it suitable for securing data on power-limited devices. Meanwhile, TinyJAMBU-192 and TinyJAMBU-256 modes require an encryption key size of 192 bits and 256 bits respectively. Due to their large key sizes, they often consume extra processing time as compared to TinyJAMBU-128. Therefore, in this research, we have only considered TinyJAMBU-128 during testing. Another alternative for improving processing time is to lower the number of rounds require for updating TinyJAMBU's internal state. Although lowering the number of permutation rounds might improve the algorithm's processing time, it could impact the resiliency of the system against data breaches. As this modification required further security analysis which we will be considering during our future works. Yet to our knowledge, our work provides a base ground for exploring a lightweight authenticated encryption scheme for securing data over CAN bus network with data traceability support and full resiliency against CAN message replay attacks.

In this article, a modified version of TinyJAMBU-128 was fully implemented and deployed on multiple CAN-enabled development boards. Data communicated over the CAN bus were encrypted and authenticated using TinyJAMBU-128. We have analyzed the performance of TinyJAMBU-128 in terms of CAN bus load, processing time, and average message communication overheads and compared the results against traditional cryptosystems, such as AES, ARIA, and Camellia. Message authentication on the classical cryptosystems was established via the employment of SHA-512. Based on our simulation, TinyJAMBU-128 outperformed AES, ARIA, and Camellia with respect to CAN bus load and average CAN data frame overhead. Since only one data frame is required to authenticate a data block. In contrast, cryptosystems that relied on SHA-512 for data authentication took 8 CAN data frames. Finally, an anti-replay attack model was fully developed and deployed on the proposed testbed. The proposed model was validated based on various injection rates. We were able to achieve a TPR between 0.99 and 0.97 for replay attack injection rates between 2% and 20% respectively. As more replay messages injected on the bus, TRP decreases to 0.92 with 50% of the transmitted messages were considered malicious.

Acknowledgements

The authors are thankful to the Deanship of Scientific Research at Najran University for funding this work, under the Research Groups Funding program grant code (NU/RG/SERC/12/27). The work of K.-K. R. Choo is supported by NSF (National Science Foundation) CREST Grant HRD-1736209, and the Cloud Technology Endowed Professorship.

References

[1] S. Hartzell, C. Stubel, and T. Bonaci, "Security analysis of an automobile controller area network bus," *IEEE Potentials*, vol. 39, no. 3, pp. 19–24, May/Jun. 2020.

[2] Y. Zhang, T. Liu, H. Zhao, and C. Ma, "Risk analysis of CAN bus and ethernet communication security for intelligent connected vehicles," in *Proc. IEEE Int. Conf. Artif. Intell. Ind. Des* ., 2021, pp. 291–295.

[3] Steve Corriagan, "Introduction to controller area network (CAN)," May 24, 2023. [Online]. Available: https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1678332963032&ref_url=https%253A%252F%252Fwww.google.com%252F

[4] H. K. Kalutarage, M. O. Al-Kadri, M. Cheah, and G. Madzudzo, "Context-aware anomaly detector for monitoring cyber attacks on automotive CAN bus," in *Proc. 3rd ACM Comput. Sci. Cars Symp.*, 2019, pp. 1–8.

[5] L. Kang and H. Shen, "Detection and mitigation of sensor and CAN bus attacks in vehicle anti-lock braking systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 6, no. 1, Jan. 2022, Art. no. 9.

[6] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection," in *Proc. 12th Annu. Conf. Cyber Inf. Secur. Res.*, 2017, pp. 1–4.

[7] D. Caivano, M. De Vincentiis, F. Nitti, and A. Pal, "Quantum optimization for fast CAN bus intrusion detection," in *Proc. 1st Int. Workshop Quantum Program. Softw. Eng.*, 2022, pp. 15–18.

[8] Q. Zhao, M. Chen, Z. Gu, S. Luan, H. Zeng, and S. Chakrabory, "CAN bus intrusion detection based on auxiliary classifier GAN and out-of-distribution detection," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 4, Jul. 2022, Art. no. 45.

[9] R. Gundu and M. Maleki, "Securing CAN bus in connected and autonomous vehicles using supervised machine learning approaches," in *Proc. IEEE Int. Conf. Electro Inf. Technol* ., 2022, pp. 042–046.

[10] Y. Qiu, T. Misu, and C. Busso, "Driving anomaly detection with conditional generative adversarial network using physiological and CAN-bus data," in *Proc. Int. Conf. Multimodal Interaction*, 2019, pp. 164–173.

[11] K. Pawelec, R. A. Bridges, and F. L. Combs, "Towards a CAN IDS based on a neural network data field predictor," in *Proc. ACM Workshop Automot. Cybersecurity*, 2019, pp. 31–34.

[12] T. C. M. Dönmez, "Anomaly detection in vehicular CAN bus using message identifier sequences," *IEEE Access*, vol. 9, pp. 136243–136252, 2021.

[13] A. Wang et al., "Anomaly information detection and fault tolerance control method for CAN-FD bus network," in *Proc. IEEE 19th Int. SoC Des. Conf* ., 2022, pp. 308–309.

[14] F. Fenzl, R. Rieke, and A. Dominik, "In-vehicle detection of targeted CAN bus attacks," in *Proc. 16th Int. Conf. Availability, Rel., Secur.*, 2021, Art. no. 32.

[15] R. Brown, A. Marti, C. Jenkins, and S. Shannigrahi, "Dynamic address validation array (DAVA): A moving target defense protocol for CAN bus," in *Proc. 7th ACM Workshop Moving Target Defense*, 2020, pp. 11–19.
[16] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "LiBrA-CAN: Lightweight broadcast authentication for controller area networks," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 3, Aug. 2017, Art. no. 90.
[17] Y. Xiao, S. Shi, N. Zhang, W. Lou, and Y. T. Hou, "Session key distribution made practical for CAN and CAN-FD message authentication," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2020, pp. 681–693.
[18] M. Zhang, P. Parsch, H. Hoffmann, and A. Masrur, "Analyzing CAN's timing under periodically authenticated encryption," in *Proc. IEEE Conf. Exhib. Des., Automat. Test Europe*, 2022, pp. 620–623.
[19] P. Liu, Y. Liu, X. Wang, C. Fang, X. Guan, and T. Liu, "Channel-state-based fingerprinting against physical access attack in industrial field bus network," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9557–9573, Jun. 2022.
[20] T. Chong, T. Liu, Y. Zhang, C. Ma, X. Jia, and Z. Wu, "Analysis of the influence of CAN bus encryption and decryption on real-time performance," in *Proc. IEEE 2nd Int. Conf. Comput. Commun. Netw. Secur.*, 2021, pp. 38–44.
[21] P. Biondi, G. Bella, G. Costantino, and I. Matteucci, "Implementing CAN bus security by TOUCAN," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2019, pp. 399–400.
[22] L. Kang and H. Shen, "Attack detection and mitigation for sensor and CAN bus attacks in vehicle anti-lock braking systems," in *Proc. IEEE 29th Int. Conf. Comput. Commun. Netw.*, 2020, pp. 1–9.
[23] P. Thirumavalavasethurayar and T. Ravi, "Implementation of replay attack in controller area network bus using universal verification methodology," in *Proc. IEEE Int. Conf. Artif. Intell. Smart Syst .*, 2021, pp. 1142–1146.
[24] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *Proc. IEEE Intell. Veh. Symp.*, pp. 1577–1583, Jun. 2017.
[25] Y. Laarouchi, M. Kaâniche, V. Nicomette, I. Studnia, and E. Alata, "A language-based intrusion detection approach for automotive embedded networks," *Int. J. Embedded Syst.*, vol. 10, no. 1, pp. 1–12, 2018.
[26] S. Katragadda, P. J. Darby, A. Roche, and R. Gottumukkala, "Detecting low-rate replay-based injection attacks on in-vehicle networks," *IEEE Access*, vol. 8, pp. 54979–54993, 2020.
[27] M. Jedh, L. Ben Othmane, N. Ahmed, and B. Bhargava, "Detection of message injection attacks onto the CAN bus using similarities of successive messages-sequence graphs," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4133–4146, 2021.
[28] B. Shannon, S. Etikala, Y. Gui, A. S. Siddiqui, and F. Saqib, "Blockchain based distributed key provisioning and secure communication over CAN FD," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2019, pp. 638–644.
[29] T. Chong, T. Liu, Y. Zhang, C. Ma, X. Jia, and Z. Wu, "Analysis of the influence of CAN bus encryption and decryption on real time performance," in *Proc. IEEE 2nd Int. Conf. Comput. Commun. Netw. Secur.*, 2021, pp. 38–44.
[30] T. Dee and A. Tyagi, "Message integrity and authenticity in secure CAN," *IEEE Consum. Electron. Mag.*, vol. 10, no. 5, pp. 33–40, Sep. 2021.
[31] F. Páez and H. Kaschel, "A proposal for data authentication, data integrity, and replay attack rejection for the LIN bus," in *Proc. IEEE CHILEAN Conf. Elect., Electron. Eng., Inf. Commun. Technol.*, 2021, pp. 1–7.
[32] TinyJAMBU v2 Specification (nist.gov), May 17, 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf
[33] Z. Kang, J. Li, J. Shen, J. Han, Y. Zuo, and Y. Zhang, "TFS-ABS: Traceable and forward-secure attribute-based signature scheme with constant-size," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 9514–9530, Sep. 2023.
[34] Y. Lu and J. Li, "Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4397–4409, Dec. 2022.
[35] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 553–567.
[36] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 4, pp. 708–722, Jul./Aug. 2018.
[37] D. Wang, W. Li, and P. Wang, "Measuring two-factor authentication schemes for real-time data access in industrial wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 9, pp. 4081–4092, Sep. 2018.
[38] Q. Jiang, N. Zhang, J. Ni, J. Ma, X. Ma, and K.-K. R. Choo, "Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 9390–9401, Sep. 2020.
[39] C. Wang, D. Wang, Y. Duan, and X. Tao, "Secure and lightweight user authentication scheme for cloud-assisted Internet of Things," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 2961–2976, 2023.
[40] Q. Wang, D. Wang, C. Cheng, and D. He, "Quantum2FA: Efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 193–208, Jan./Feb. 2023.
[41] D. Wang, D. He, P. Wang, and C.-H. Chu, "Anonymous two-factor authentication in distributed systems: Certain goals are beyond attainment," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 428–442, Jul./Aug. 2015.
[42] Q. Wang and D. Wang, "Understanding failures in security proofs of multi-factor authentication for mobile devices," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 597–612, 2023.
[43] N. Koblitz and A. Menezes, "Another look at 'provable security," *J. Cryptology*, vol. 20, pp. 3–37, 2007.
[44] M. Dworkin et al., "Advanced encryption standard (AES)," Federal Inf. Process. Standard (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, USA, 2001, doi: 10.6028/NIST.FIPS.197.
[45] D. Kwon et al., "New block cipher: ARIA," in *Proc. Inf. Secur. Cryptology - ICISC 2003*, 2004, pp. 432–445.
[46] K. Aoki et al., "*Camellia*: A 128-bit block cipher suitable for multiple platforms — Design and analysis," in *Proc. Sel. Areas Cryptography. SAC 2000*, 2001, pp. 39–56.
[47] stm.com, Arm Cortex-M4 32b MCU+FPU, 125 DMIPS, 512KB Flash, 128KB RAM, USB OTG FS, 11 TIMs, 1 ADC, 13 comm, 2017. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f411re.pdf
[48] Arm Mbed OS, 2023. [Online]. Available: https://os.mbed.com/mbed-os/
[49] Mbed TLS, 2023. [Online]. Available: https://os.mbed.com/docs/mbed-os/v6.16/apis/tls.html
[50] A. Rex, R. Amar, V. Hacer, B. Mohamed, M.-S. Louanne, and R. Mahapatra, "Harnessing IoT technology for the development of wearable contact tracing solutions," in *Proc. TRON Symp. (TRONSHOW)*, 2021, pp. 1–9.
[51] R. Clarke, L. McGuire, M. Baza, A. Rasheed, and M. Alsabaan, "Online voting scheme using IBM cloud-based hyperledger fabric with privacy-preservation," *Appl. Sci.*, vol. 13, no. 13, 2023, Art. no. 7905.

Amar Rasheed is currently an Assistant Professor with the Department of Computer Science, Sam Houston State University, Huntsville, TX, USA. He was a Postdoctoral Fellow with the Information Science and Technology Division of the Applied Research Laboratory, Pennsylvania State University, State College, PA, USA. His research interests include sensor modeling and data collection algorithms, efficient data collection schemes for wireless sensor networks, energy-efficient sensor data gathering mechanisms, secure mobile sensor data communication models design, cybersecurity systems, cybersecurity risk assessment, and analysis.

Mohamed Baza received the Ph.D. degree from Tennessee Technological University, Cookeville, TN, USA, in December, 2020. He is currently an Assistant Professor with the Department of Computer Science, College of Charleston, Charleston, SC, USA. He is the author of many journals and conferences, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, ICC, and CCNC. His research interests include blockchains, cybersecurity, and machine learning.

Mahmoud. M. Badr received the B.S. and M.S. degrees in electrical engineering (electronics and communications) from Benha University, Cairo, Egypt, in 2013 and 2018, respectively, and the Ph.D. degree in electrical and computer engineering from Tennessee Tech University, TN, USA, in 2022. He is with the Department of Network and Computer Security, College of Engineering, SUNY Polytechnic Institute, Utica, NY, USA. His research interests include machine learning, blockchain, cryptography, 5G networks, and smart grids. He has been selected as a poster winner at Tennessee Tech. University's annual Research and creative inquiry day, 2021.

Hani Alshahrani received the bachelor's degree in computer science from King Khaled University, Abha, Saudi Arabia, the master's degree in computer science from California Lutheran University, Thousand Oaks, CA, USA, and the Ph.D. degree from Oakland University, Rochester, MI, USA. He is currently an Associate Professor of computer science and information systems with Najran University, Najran, Saudi Arabia. His research interests include smartphones, IoT, crowdsourcing security, and privacy.

Kim-Kwang Raymond Choo (Senior Member, IEEE) received the Ph.D. degree in information security from the Queensland University of Technology, Brisbane, QLD, Australia, in 2006. He currently holds the Cloud Technology Endowed Professorship with The University of Texas at San Antonio, San Antonio, TX, USA. He is the founding co-Editor-in-Chief of ACM Distributed Ledger Technologies: Research & Practice, and the founding Chair of IEEE Technology and Engineering Management Society Technical Committee (TC) on Blockchain and Distributed Ledger Technologies. He was the recipient of the 2022 IEEE Hyper-Intelligence TC Award for Excellence in Hyper-Intelligence Systems (Technical Achievement award), 2022 IEEE TC on Homeland Security Research and Innovation Award, 2022 IEEE TC on Secure and Dependable Measurement Mid-Career Award, and the 2019 IEEE TC on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher).