ShuffleCAN: Enabling Moving Target Defense for Attack Mitigation on Automotive CAN

Huiping Qian*, Hao Han*, Xiaojun Zhu*, Fengyuan Xu†
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
†State Key Laboratory for Novel Technology, Nanjing University, China

***Abstract*—Controller Area Networks (CANs), the most widely used protocols for in-vehicle networks, are vulnerable to various attacks due to the lack of security countermeasures by design. CAN messages are broadcast without source/destination labeling and lack built-in encryption or authentication mechanisms, thus suffering many attacks. To address this problem, we propose a lightweight CAN message obfuscation technique called ShuffleCAN. Motivated by the idea of moving target defense (MTD), ShuffleCAN is designed with a combined shuffling scheme based on the hash chain and combinatorial coding techniques to achieve both ID anonymization and payload shuffling. With ShuffleCAN, selected or all transmitter and receiver pairs can communicate in a private dialect over the standard CAN protocol, so the eavesdropper cannot understand the meaning of each message or inject a valid fake message. We implemented a prototype and evaluated ShuffleCAN on Toyota's testbed PASTA. The experimental results show that ShuffleCAN outperforms state-of-the-art CAN protection schemes.**

***Index Terms*—Automotive Security, Controller area network, Anonymous ID, Shuffle**

I. Introduction

Controller Area Networks (CANs) were developed in the early 1980s and widely utilized in various fields, including automotive, marine, and industrial control systems. Due to benefits such as robustness and low cost, CAN has become the *de facto* communication standard for in-vehicle networks since 2008. Over CANs, various Electronic Control Units (ECUs) can acquire and interpret sensor readings and adjust actuators to control nearly all vehicle functions, including steering, acceleration, lighting, etc. Unfortunately, CANs are primarily designed to ensure reliable communication, but not for security. The CAN bus broadcasts messages without source/destination labeling and lacks encryption, authentication, or other security protection commonly used in IT networks. Thus, an adversary can easily eavesdrop on CAN messages to reverse engineer desired data signals or inject fake CAN messages with spoofed IDs or content. This may lead to an invasion of confidentiality, integrity, and availability. Researchers have already demonstrated various attacks through CANs [5], [8], [13].

For decades, numerous schemes have been proposed to secure CAN communication. There exist some encryption mechanisms [1], [2], [4] as well as hardware acceleration for cryptography, but automotive stakeholders do not widely adopt them. Hardware-based encryption requires modification to existing ECUs' hardware, which means deployment is more expensive. A software solution is preferable with better compatibility but may cause extra latency on resource-constrained ECUs, which is unacceptable for safety-critical real-time systems. In fact, the encryption of the entire CAN messages is unnecessary since only a few bits contain meaningful information (see Section II for the detail). Another method of securing the CAN bus is to add authentication, such as using Message Authentication Codes (MACs). However, since the data field in a CAN message is confined to 8 bytes, only truncated MAC can be used. Hence, existing solutions have not yet been sufficient to protect the CAN.

To address the problem, this paper proposes a lightweight CAN message obfuscation scheme dubbed ShuffleCAN. Motivated by the idea of Moving Target Defense (MTD), ShuffleCAN shuffles both the ID and bit position of the message payload in a chain so that only the sender and receivers know the changing sequence. By design, ShuffleCAN does not waste time encrypting the message content. Instead, only the bit positions of 0 and 1 are exchanged in a message. We believe unbiased shuffling is sufficient to harden various attacks over CAN. It should be noted that ShuffleCAN is designed to be lightweight and complementary to existing data ciphers. It should not incur too much burden on ECUs and should cost minimal software modification.

However, designing an efficient shuffling scheme is not straightforward. The first challenge is reducing the computation cost for running the shuffling algorithm on resource-constrained ECUs. Traditional shuffling algorithms such as Fisher–Yates [12] are able to generate unbiased permutation sequence, but it is not efficient for CAN. We argue that exchanging two bits with the same value (0 to 0 or 1 to 1) is meaningless. To this end, we propose a novel shuffling algorithm based on combinatorial coding and lexicographic ordering techniques. Secondly, it is unclear how to ensure that the sender and receivers are synchronized well when the transmission fails. For example, a certain ECU occurs bus-off, while the sender may communicate with other receivers normally. Even though the ECU resets, it will never understand the current shuffled messages. We propose an error detection and recovery algorithm for the failed receiver to "catch up".

We have implemented and evaluated ShuffleCAN on Toyota's CAN testbeds. Our evaluation results show that ShuffleCAN can achieve not only recoverable ID anonymization in the preservation of the frame priority but also efficient unbiased data shuffling.

Hao Han (hhan@nuaa.edu.cn) is the corresponding author.

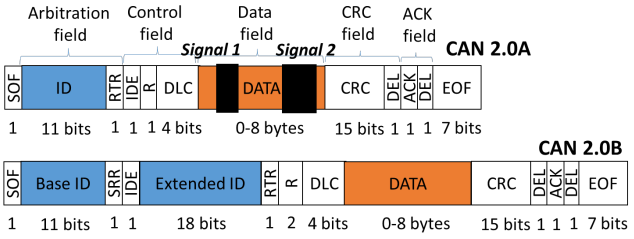

Fig. 1. Format of CAN frames defined in CAN 2.0A and CAN 2.0B

This paper makes the following main contributions:

- Proposal of a lightweight CAN message obfuscation scheme by means of shuffled data payload, motivated by the idea of moving target defense (MTD). A novel unbiased shuffling algorithm based on combinatorial coding and lexicographic ordering is proposed to achieve synchronized coding/decoding without exchanging information between the sender and receivers.
- Design of a hash-chain-based error detection and recovery mechanism that supports one-to-many communication over CAN. Such a mechanism enhances existing error-handling schemes specified in CAN to tolerate message loss and ECU failure.
- Implementation and evaluation of ShuffleCAN in the real testbed, consisting of 4 Renesas ECUs. The experimental results show that our approach is compatible with standard CAN bus protocols and performs better than existing schemes.

The rest of this paper is organized as follows: Section II presents the background and related work. Section III discusses the adversary model and makes some assumptions. Section IV shows the design rationale and provides the details of ShuffleCAN. Section V discusses evaluation results, and Section VI concludes the paper.

II. Background and Related Work

A. CAN Bus

Controller Area Network (CAN), standardized in ISO 11898 [24], is one of the most widely deployed in-vehicle networks. CAN is a serial, multi-master bus designed for reliable, real-time message delivery between ECUs.

CAN Message Format: Fig. 1 shows the format of CAN frames. The first bit of a CAN frame is an indicator of the start of frame (SOF), followed by the arbitration field, which contains either an 11-bit identifier (ID) defined in CAN 2.0A or a 29-bit identifier defined in CAN 2.0B. The arbitration field also includes a single Remote Transmission Request (RTR). Next, the control field has a 4-bit data length code (DLC). The data field varies from 0 to 64 bits (8 bytes), which may contain several CAN signals. The CAN frame with a specific CAN ID carries 2 CAN signals in this figure. Last, there are fields for CRC and ACK.

Message Transmission: CAN frames are transmitted in a broadcast way. All connected ECUs can listen to other communications or write to the entire bus. CAN messages do not identify the sender or designated receiver, only the message ID. This violates the security design principle of complete mediation. There is no authentication of source messages, no mechanism to verify the integrity of a message, and no system to verify that a node has the authority to send or receive a given message. Furthermore, a priority-based carrier sense multiple access (CSMA) arbitration scheme is used in case multiple nodes try to transmit frames simultaneously. If a node tries to transmit a frame and senses the bus is busy, it has to wait until the bus is free. If two nodes accidentally send packets simultaneously, the frame with the highest priority (i.e., the smallest ID) will win the arbitration. Thus, an attacker may easily inject frames with a small ID to launch DoS attacks.

B. Related Work

Recent research has demonstrated that attacks against the CAN are becoming prevalent. Existing countermeasures can be broadly classified into three categories: 1) ID randomization, 2) data protection, and 3) intrusion detection.

The goal of ID randomization is to protect CAN against certain attack types such as spoofing. As pioneering works, IA-CAN [15] is proposed to enable ID randomization. However, it has no concern with data protection and multiple receivers. Recent studies consider leveraging the idea of MTD to achieve advanced ID protection [6], [22], [26]. A CAN ID shuffling technique named CIST [26] is proposed. However, it only supports peer-to-peer communication, as well as the presence of a high burden of AES encryption. CANsafe [22] defends against DoS attacks by adjusting the priority rules for different IDs dynamically while requiring additional hardware modification.

Another way of protection is to use their data fields for encryption [25] or authentication [27] [11]. Lin et al. [18] proposed a Message Authentication Code with pre-assigned paired keys. Such schemes may require sending an additional frame carrying MAC information or embedding authentication in the data field, thus compressing the message capacity. Also, Lu et al. [19] introduced a scheme named LEAP, exploiting the security-enhanced RC4 stream cipher primitives.

In addition to the above research on CAN frames, existing research on in-vehicle networks includes intrusion detection. One is to study the fingerprint of the vehicle device. Each ECU has some unique characteristics, such as voltage. Kang et al. [9] proposed Viden, using the voltage distribution of the ECU as its specific fingerprint. Another approach is to use deep learning schemes such as RNN and LSTM [16].

To summarize, ShuffleCAN differs in that we leverage the idea of MTD to shuffle data bits rather than the ID field only. We believe that traditional encryption is inefficient because many bits in CAN messages do not carry any useful information, and we propose a shuffling scheme based on combinatorial mathematical coding and decoding to improve efficiency.

III. Adversary Model and Assumptions

We assume that the adversary is capable of accessing the CAN bus via numerous attack vectors. Through physical

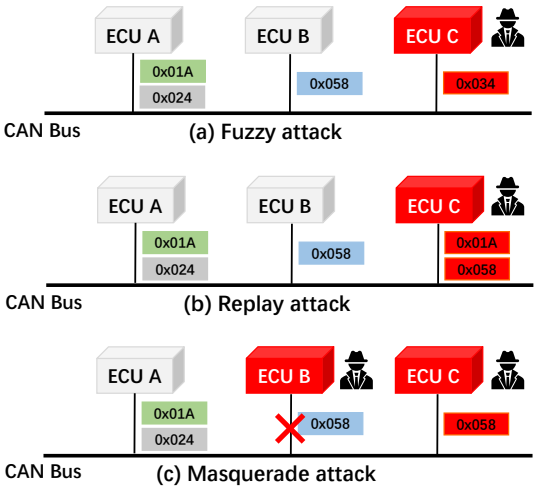

Fig. 2. Three main attack scenarios on the CAN, including fuzzy, replay, and masquerade attacks.

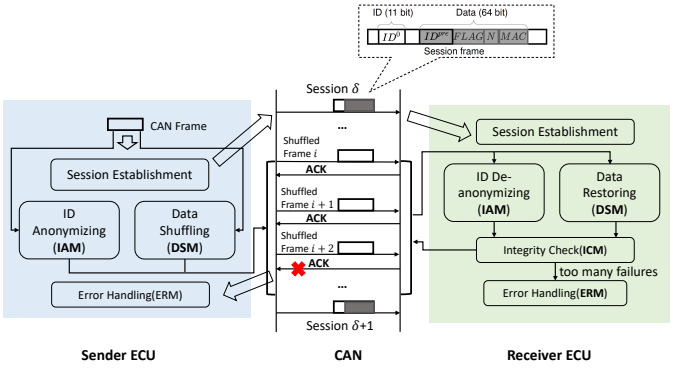

Fig. 3. Overall workflow of ShuffleCAN system

access and remote access, such as plugging into On-Board Diagnostics II (OBD-II) [7], hacking into a connected ECU via Bluetooth or Wi-Fi [20], an attacker can eavesdrop on CAN message transmitted through CAN, compromise a specific ECU, or inject forged CAN message into the bus. Through USB or Wi-Fi access, the adversary can compromise and manipulate non-critical ECUs such as infotainment units to gain control of the CAN and send arbitrary frames to confuse the communication.

Based on the attack vectors mentioned above, we broadly classify the CAN attacks into three categories: *fuzzy attack*, *replay attack*, and *masquerade attack*. As shown in Fig. 2(a), a fuzzy attack injects fake messages of random ID and payload to infer the control domain and parsing format of related IDs and data. Replay attack in Fig. 2(b) requires that the adversary first records CAN messages and re-sends them later. Fig. 2(c) illustrates the idea of a masquerade attack, which requires compromising two ECUs at least. This attack needs to suspend the legitimate ECU using a weak attack to prevent the ECU from sending frames and then use a strong attack to compromise another ECU to send specific frames, e.g., the frames that the last corrupted ECU would have sent.

In addition, we assume that each ECU pre-shares long-term symmetric keys matching the ID with other ECUs. To the best of our knowledge, this is the mechanism used by most security protection schemes for CAN bus [15] [26] [25] [21] [14]. In the work of [25] [21], they assume that each ECU stores the corresponding long-term symmetric keys individually, while the gateway ECU stores all the keys matching the ECU. Here we do not need a gateway ECU for multiple session key distributions; instead, we deploy the authentication scheme to each ECU entity and bind it to its IDs. All symmetric keys are configured and written to the ECU at assembly time, and it is assumed that this storage will not be compromised by the adversary.

IV. ShuffleCAN Design

In this section, we detail several parts of ShuffleCAN, including 1) the session initialization and update process, 2) how IDs are randomly anonymized, 3) the dynamic bit position shuffling scheme based on combinatorial mathematics, and 4) how error recovery is performed if necessary.

A. System Overview

Fig. 3 shows the overall workflow of the proposed ShuffleCAN system. Before transmitting any CAN frame, the sender determines whether to start a new session. If so, the sender will broadcast a session frame to receivers. In the session, all CAN frames are dynamically translated into a special "dialect" that is only known between the sender and the receiver(s). In particular, the ID field of the frame is anonymized by **ID Anonymizing Module** ($\mathcal{IAM}$), and the data field is shuffled by **Data Shuffling Module** ($\mathcal{DSM}$). When the "dialect" frame is received by receivers as they expect, the receivers use $\mathcal{DSM}$ to restore the data content into the original format, and then feed into the **Integrity Check Module** ($\mathcal{ICM}$) for counterfeit detection. This module includes the existing fault confinement mechanism defined in the standard of CAN protocols.

B. Session Initialization

The goal of session initialization is to establish a session key between a sender and the receiver(s), as well as share session parameters. The session protocol is one-way, meaning that only the sender broadcasts session frames. Similar to existing works such as [15], we assume that every frame ID_i is associated with a unique secret key k_i as mentioned in Section III. For instance, ECU_1 is designed to receive ID_2 and ID_3 both sent by ECU_2, and ID_4 sent by ECU_3. Thus, both ECU_1 and ECU_2 should know k_2 and k_3 associated to ID_2 and ID_3, respectively. ECU_1 and ECU_3 need to know the key k_4 for communication.

To start a session, the sender first broadcasts a session frame shown in Fig. 3, where ID^0 denotes the origin ID; ID^{pre} is

the ID used in the prior session. If the upcoming session is the first one, $ID^{pre} = ID^0$. $FLAG$ includes session parameters, including the maximum number of frames, the frequency to update a new hash value, etc. N is a nonce generated by the sender for each session. The MAC is calculated by Eq. (1),

$$MAC = f_1(ID^{pre}, N, FLAG, k_i), \tag{1}$$

where f_1 is a keyed hash function.

Upon receiving the session frame, the receiver interested in ID^0 frames uses the same equation above with the carry-on values of N and ID^{pre}, coupled with the secret k_i to compute MAC. Once two values are matched, the receiver confirms that this session frame is valid. Due to the use of ID^{pre}, N, and MAC fields, our protocol is protected against spoofing replaying attacks. Otherwise, the receiver will issue an active error flag to notify the failure of authentication. The success of this method is based on the fact that k_i will never be shared over CAN and thus unknown to an attacker. After that, the receiver is ready to receive the upcoming frames in the session with the session key K_δ by Eq. (2),

$$K_\delta = f_1(N, k_i, \delta), \tag{2}$$

where δ refers to the session number.

When reaching the maximum number of frames, the sender will start a new session by providing new ID^{pre}, N, MAC, and $FLAG$. The lifetime of a session is generally the startup time of vehicles. It is free for a sender to terminate a session at any time. In order to prevent the replay attack, the consequent session frame includes the keyed hash value of the last anonymized ID used in the previous session rather than ID^0.

C. Anonymizing and Deanonymizing Frame IDs

The goal of **ID Anonymizing Module** ($\mathcal{IAM}$) is to prevent the attacker from tracking important CAN frames. We adopt the idea of the hash chain from work [15]. In each session, the ID of every m frame (by default $m = 1$) changes dynamically using a keyed hash chain. The key difference between ShuffleCAN and work [15] is in three aspects: 1) ShuffleCAN uses new keys derived from the session key, 2) protects the data content, and 3) introduces a mechanism to handle transmission errors.

Before digging into the details, we first clarify some terms:

- *Session key (K_δ):* Derived in the session initialization phase and never used to anonymize frame IDs directly.
- *Priority ID (PID):* Static part of the anonymized ID to preserve the priority of the original ID. Recall that a smaller ID wins the arbitration. What we need is $PID_i < PID_j$ given that $ID_i < ID_j$.
- *Dynamic ID (DID):* Dynamic part of the anonymized ID used only in hash-chain computation.
- *Anonymous ID (AID):* Truncated DID' that is actually transmitted over CAN along with PID.
- *Anonymous Key (AKey):* Derived from the session key K_δ and changed with each data frame communication. The formula expression is K_δ^n.

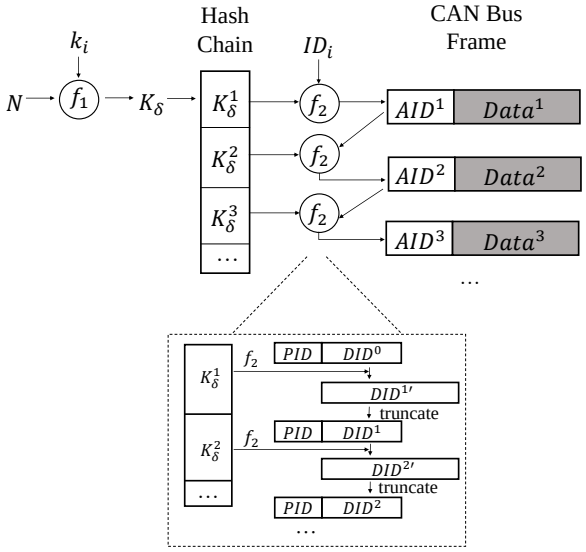

Fig. 4. Illustration of the process of $\mathcal{IAM}$ for ID anonymization

On the sender side, given K_δ derived during the session initialization, the sender calculates the hash of that key to obtain K_δ^1 for the first frame as shown in Fig. 4. The original ID (i.e., ID_i) is divided into two parts: PID and DID. SAE J1939 standard assigns the first 3 bits of the 29-bit ID for specifying priority [15]. Then, K_δ^1 and AID with padding are hashed to generate DID^1 by Eq. (3),

$$DID^{n'} = f_2(AID^{n-1}, K_\delta^n). \tag{3}$$

Since the size of $DID^{1'}$ is larger than the ID field, $DID^{1'}$ will be truncated. Next, the sender concatenates PID and DID^1 (i.e., truncated $DID^{1'}$) and fills it into the ID field of the first frame in this session. In order to avoid possible ID conflicts when the scheme is only applied to part of ECUs, an ID pool is used to exclude ID frames sent by normal ECUs to avoid ID collisions. For any succeeding frame (saying, frame i), the anonymizing key K_δ^i will be hashed from its previous key K_δ^{i-1}, and the sender repeats the process to calculate AID^i with AID^{i-1} and K_δ^i. It should be noticed that if $m > 1$, more than one frame will use the same anonymous key.

On the receiver side, the deanonymizing process determines whether the received (anonymized) frame is desired. Similar to the sender, the receiver derives the expected ID for the first frame based on K_δ. After that, the frame filter of the receiver is updated to accept such an ID only. Once the desired frame is received, the same process will update the filter to accept the consequent frames one by one in the hash chain.

Error Recovery Module ($\mathcal{ERM}$): When a certain count threshold is exceeded, the error recovery mechanism is activated to help the bus-off ECUs recalculate the anonymous ID and parse the shuffled data. The main idea of this scheme is to calculate the expected ID value and configure the frame filter to receive all frames. Due to space limitations, we cannot give too much about error recovery methods here. Actually, our

scheme can implement mechanisms such as reliable session reconnection.

D. Shuffling and Restoring Data Bits

Data Shuffling Module ($\mathcal{DSM}$) is designed to shuffle the data field of each frame dynamically. The shuffling process relies on the same key used in $\mathcal{IAM}$. Again, our goal is to achieve a lightweight permutation to change the distribution of the original data content for any length of data "bins".

Strawman 1: Our first attempt at data shuffling is to use the Fisher-Yates shuffle algorithm [12]. Given the size of the data field equals 64, the steps are as follows:

1) Pushing all bits of the data field into an array of 64 elements (indices $[0, 63]$);
2) Staring from the last element down to 1, picking a random number j such that $1 \leq j \leq n-1$ and exchanging the last element and element j;
3) Striking out the last element and shrinking the array to contain the remaining elements;
4) Repeat step 2 until only one element is left in the array;
5) The sequence is a permutation of the initial elements.

It has already been proved that the Fisher-Yates algorithm can yield an unbiased permutation within $O(n)$ given that the random number picked in step 2 is genuinely random. The required storage is $O(1)$. However, the shortcomings are obvious: 1) the algorithm requires n total random numbers, and 2) the algorithm cannot be accelerated in parallel. Hence, the Fisher-Yates is not suitable for resource-constrained ECUs.

Strawman 2: To further improve efficiency, we utilize the fact that the sequence considered in our scenario only has values 0 or 1. Hence, a look-up table is created to store all permutations beforehand. Each row in the table contains the combinations of n-choose-k bits. Given the number of '1' bits, the sender and receiver(s) can look up the same row and use the same random offset for shuffling. The advantage of this approach is $O(1)$ computation complexity but requires $O(n!)$ space. Although we can reduce the space by half by saving up to 32 bits in the table, the required space is still too large.

Final Design: Due to the space complexity, the look-up table approach does not suit resource-constrained ECUs. If all combinations can be arranged in a special order, there is no need to save. Motivated by this, we adopt the tool of combinatorial coding and lexicographic ordering proposed in [17]. All possible combinations are organized into a lexicographic order implying a ranking from smallest to largest, while the combinatorial encoding generates an index for the selected item in the order, and the decoding part is to produce the corresponding selection of items. With this design, ShuffleCAN can achieve unbiased data shuffling with both small computation and space overhead.

We first present the encoding-decoding algorithm and propose a shuffle scheme by the introduction of AKey. Given a bit string s with a length of N. The following formula denotes an ascending array:

$$I_s = [I_s(1), ..., I_s(K)], \tag{4}$$

Algorithm 1: Encode Function

Input: Ordered I_s, N, K
Output: $index_s$

```
if I_s[0] < 0 or len(I_s) > N then
    error;
end
index ← 0;
for k ← 1 to K do
    n ← I_s(k);
    if n ≥ k then
        index_s ← index_s + C(n, k)
    end
end
```

where K represents the number of '1' in the string, and $I_s(i)$ indicates the position of the ith '1' bit with $i \in [1, K]$ and $I_s(k) \in [0, N-1]$. Eq. (5) converts the string s into an $index_s$ in the lexicographic order of combinations of N choose K:

$$index_s = \sum_{k=1}^{K} \binom{I_s(k)}{k}. \tag{5}$$

Alg. 1 shows the encoding process, i.e., mapping a sequence to an index.

The decoder reverses the combined index $index_s$ to the string s. Alg. 2 shows the decoding process. We maintain variable $index_s$ and reduce it once a '1' bit is determined. Variables n and K represent the length of the string and the number of '1', initialized to $N-1$ and the input K. In each iteration, we compare a binomial coefficient, $\binom{n}{K}$, with the current value of $index_s$. If $index_s$ is greater, the K-th location is set as '1', followed by subtracting the binomial coefficient from $index_s$. The process terminates either n or K reaches 0.

For example, suppose the received string $s = 1001001011$, we obtain $I_s = [0, 3, 6, 8, 9]$ with $K = 5$ and $N = 10$. Through Eq. (5), we have $index_s = \binom{0}{1} + \binom{3}{2} + \binom{6}{3} + \binom{8}{4} + \binom{9}{5} = 219$, with the define of $\binom{0}{1} = 0$. To decode string s, we repeat comparing the remaining value of $index_s$ to $\binom{N-1}{K}$. Initially, 219 is greater than $\binom{9}{5}$, so the 9th position of the decoded string is '1'. Next, subtract $\binom{9}{5}$ from $index_s$ which remains 93 now. Meanwhile, we decrease N and K by 1, resulting in $N = 8$ and $K = 4$. Once again, 93 is greater than $\binom{8}{4}$, so the 8th position of the string is still '1'. The remaining $index_s$ become 23, $N-1$ is 7, and $K-1$ is 3. This time 23, is smaller than $\binom{7}{3}$, which means the 7th position is '0'. The above procedure is repeated until $index_s = 0$.

With encoding and decoding algorithms, we reuse the derived AKey. The workflow of the sender is as follows: with the original data as an input, the sender generates the $index_s$ using Alg. 1. Next, $index_s$ is added by K_δ^n:

$$index'_s = (index_s + K_\delta^n) \mod \binom{N}{K}, \tag{6}$$

yielding a new index. Then, the sender uses Alg. 2 to decode $index'_s$, gets the shuffled data, and broadcasts it over the

Algorithm 2: Decode Function

Input: $index_s$, N, K
Output: I_s

```
if index_s < 0 or index_s ≥ C(n,k) then
    error;
end
for n ← N − 1 to 0 do
    node ← 0;
    if n ≥ K then
        node ← C(n,k);
    end
    if index_s ≥ node then
        I_s[K] ← n;
        if K ≤ 1 then
            break;
        end
        K ← K − 1;
        index_s ← index_s − node;
    end
end
```

CAN. When receiving such a frame, the receiver encodes its data field into an index (saying, $index'_s$), subtracts K^n_δ, and decodes it for the original data. All binomial coefficients, i.e., values that are used in the coding process, can be calculated in advance and stored in a table. In other words, we need to save an array of length $\sum_{i=1}^{64} i = 2,080$ in advance to ensure timeliness.

Integrity Check Module ($\mathcal{ICM}$): To check the integrity of shuffled data, we reuse the CRC field of the CAN. By design, the CRC field contains the checksum of original data rather than the shuffled. An adversary who does not know the key and shuffling sequence is less likely to inject valid frames.

V. Evaluation of ShuffleCAN

In this section, we implemented and evaluated ShuffleCAN on a real testbed by answering the following questions:

Q1: *Can ShuffleCAN improve real performance and reduce overhead compared to the state-of-the-art systems?*

Q2: *Whether the proposed shuffling scheme is effective in increasing the degree of data confusion?*

Q3: *What attacks can be defended if the vehicle is equipped with the ShuffleCAN?*

A. Implementation and Experiment Setup

As shown in Fig. 5, we implemented a prototype of ShuffleCAN on PASTA, an open-source automotive security testbed developed by TOYOTA. PASTA provides a CAN onboard network using real ECUs/CGWs. Each ECU uses Renesas's RX63N microcontroller with 2MB ROM, sufficient for storing our lexicographic encoding tables. To modify the functionality of each ECU, we need to reprogram the firmware using the C language and flash it through an RS-232C port. PASTA simulates normal vehicle driving and assistance functions through three ECUs (including a powertrain, chassis, and body) and a gateway ECU to forward CAN messages. Messages communicated over the CAN bus involve 45 CAN IDs, each with a corresponding transmit cycle and data parsing format. All messages are periodic, with periods ranging from 10 ms to 500 ms. The hash function used in the calculation is SHA-1. We successfully deployed our approach on the testbed to protect all four ECUs. After the deployment, all existing vehicle functions work properly.

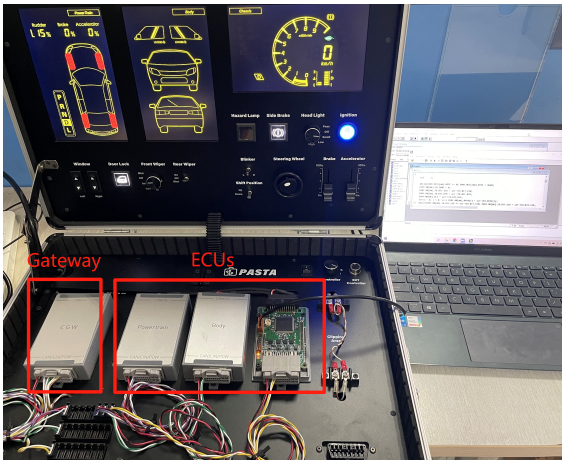

Fig. 5. Experiment setup in Toyota's PASTA testbed

B. Evaluation of Run-time Performance

The first evaluation metric for the proposed system is the overall run-time overhead. This answers Q1: how is ShuffleCAN compared to well-known shuffling algorithms and industrial standard block ciphers on real performance? We compared ShuffleCAN to six representative algorithms, namely Fisher-Yates, AES-128, RC4, BlowFish, SM4, and SPECK. AES algorithm encrypts data in CIST [26] and LEAP [19] chooses RC4 stream cipher for lightweight encryption. The source of the data is randomly generated, and the number of frames exceeds 80,000. Instead of using standard encryption codes, we rely on OpenSSL to encrypt/decrypt data for some schemes (such as AES). We choose OpenSSL over other potentially faster hardware-based solutions to ensure comparability. ShuffleCAN provides a purely software-based solution to achieve shuffling without hardware overhead. If a hardware-based solution is used, the corresponding device must be paired for ECUs. In addition to the traditional encryption scheme, we choose IoT-level symmetric encryption for comparison. SPECK is designed to meet the need for secure, flexible, and analyzable lightweight block ciphers. It offers excellent performance on hardware and software platforms and is flexible to implement on a given platform [3].

Table I shows the data processing time and memory usage for different shuffling and encryption schemes. It is seen that ShuffleCAN is $55\times$ faster and $118\times$ faster for data shuffling and restoring than Fisher-Yates. The look-up table shuffle failed due to excessive memory. Compared to industrial block ciphers, ShuffleCAN outperforms AES-128 by $2.74\times$ and $6\times$

TABLE I
COMPARISON OF DATA PROCESSING TIME, VMSIZE AND VMRSS OF DIFFERENT SHUFFLING AND ENCRYPTION SCHEMES

Scheme	Sender (ns)	Receiver (ns)	VMSize (kb)	VMRSS (kb)
ShuffleCAN	26	12	2528	612
Fisher-Yates	1428	1425	2496	580
AES-128	71	72	10588	7676
RC4	35	41	5572	1424
BlowFish	107	102	10588	7444
SM4	298	300	10588	7444
SPECK	32	35	2496	572

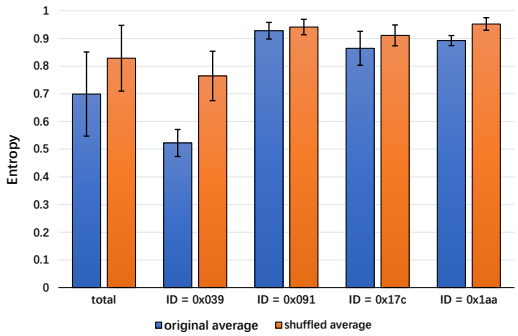

Fig. 6. Comparison of average entropy and standard deviation of consecutive data frames and different CAN-ID data frames before and after shuffling

on the sender and receiver sides. In addition, ShuffleCAN is $4.1\times$ and $8.5\times$ faster than Blowfish and $11.4\times$ and $25\times$ faster than SM4. Even compared to the IoT-level SPECK algorithm and lightweight RC4 encryption, ShuffleCAN has a corresponding time advantage, being $1.23\times$ and $2.91\times$ and $1.34\times$ and $3.41\times$ faster. The time advantage of ShuffleCAN over these schemes depends on the fact that ShuffleCAN does not waste time on changing the data content but implements lightweight shuffle-by-bit.

For memory footprint, we mainly measured two memory usages: VMSize, referring to virtual memory size, and VMRSS, which means virtual memory resides in physical memory. In the right two columns of Table I, we see that shuffling algorithms, including ours and Fisher-yates, consume less memory than conventional encryption algorithms. Although SPECK has a small or negligible memory advantage, it doesn't match ShuffleCAN in terms of time.

C. Evaluation of Randomness and Unbiasedness

The second evaluation metric is the randomness and unbiasedness of ID anonymization and data shuffling, which answered Q2 and Q3 experimentally. Here our metrics are the entropy of data and the unbiasedness of the anonymous key as well as ID. The entropy reflects the degree of confusion by ShuffleCAN, which answers Q2. Meanwhile, unbiasedness is used to indicate whether ShuffleCAN can resist attacks, etc., so as to answer Q3.

Since the shuffling process cannot capture specific bit changes, we use the entropy of the bit string instead. The CAN messages used to compare are from the real vehicle ECUs. Fig. 6 shows the comparison of the average entropy values and standard deviations of consecutive CAN data frames(e.g., total) and data frames after ID classification(e.g., ID = 0x039, 0x091, 0x17c, 0x1aa) before and after shuffling. The four IDs are selected randomly. Here we have quoted BiEntropy to calculate the entropy, which fluctuates between 0 and 1, with closer to 1 meaning more disordered. BiEntropy is based upon a weighted average of the Shannon Entropies of all, but the last binary derivative of the string [10]. For consecutive data frames, the shuffled bit string has a significant entropy improvement along with a smaller standard deviation, while for different IDs, the results are not exactly the same: for bit string frames with low original entropy, the shuffling scheme has a better effect, and for data where the original entropy is already large, shuffling can also bring some entropy improvement.

Another randomness metric evaluates ShuffleCAN's performance of anonymous. We use NIST test suits to assess anonymizing key(AKey) and anonymous ID(AID). NIST test suits are statistical test suites for random and pseudorandom number generators for cryptographic applications [23]. Once the test result is P-Value ≥ 0.01, the sequence is supposed to be random. We prepare a real vehicle data chain of lengths over 10 million for each test. Five test results are shown in Table II but all sixteen tests are passed.

D. Comparison with Existing Schemes

This subsection answers Q1 through theoretical analysis. Table III shows the comparison of ShuffleCAN and other state-of-the-art approaches including IA-CAN [15], CIST [26], DAVA [6], and LEAP [19]. While ShuffleCAN and CIST use MTD to achieve **ID randomization**, other schemes are either not anonymous or consume resources. Similarly, whether **data is protected** is another important attribute, and we experimentally confirmed that ShuffleCAN can achieve higher efficiency than encryption. Considering the broadcast nature of CAN, **supporting one-to-many** communication, i.e., multiple receiver ECUs receiving a modified frame simultaneously, is a vital quality. **No center control** is also one of the significant metrics, as a centralized ECU brings the possibility of collective ECUs being compromised if it is compromised. **Low overhead** also matters to measure feasibility. Our approach supports all capabilities with lower performance overhead.

VI. CONCLUSION

In this paper, we propose a shuffle-based data obfuscation protection scheme with id anonymity, ShuffleCAN. We first point out the static features by reviewing past works and then are motivated by moving target defense(MTD). To achieve this goal, we introduce a dynamic source as the parameter for anonymous ID($\mathcal{IAM}$) and data shuffling($\mathcal{DSM}$). By using ID chains, we subsequently propose corresponding error recovery modules($\mathcal{ERM}$) to cope with ECU errors and an integrity check module($\mathcal{ICM}$) to detect attacks such as tampering. Moreover, a novel unbiased shuffling algorithm based on

TABLE II
NIST TESTS FOR THE RANDOMNESS OF ANONYMIZING KEY AND ANONYMOUS ID

Test Name	Anonymizing key		Anonymous ID	
	P-Value	Result	P-Value	Result
Frequency	0.772760	Success	0.911413	Success
BlockFrequency	0.148094	Success	0.888137	Success
Runs	0.706149	Success	0.637119	Success
FFT	0.275709	Success	0.232760	Success
Universal	0.568055	Success	0.602458	Success

TABLE III
QUALITATIVE COMPARISON OF SHUFFLECAN TO RELATED APPROACHES

	ShuffleCAN	IA-CAN	CIST	DAVA	LEAP
ID randomization	✓	✓	✓	✓	×
Data protection	✓	×	✓	×	✓
Support one-to-many	✓	✓	×	×	×
No center control	✓	✓	×	✓	×
Low overhead	✓	×	×	×	✓

combinatorial coding and lexicographic ordering is proposed to achieve an efficient shuffling process on the resource-constrained ECUs. We implement and evaluate ShuffleCAN on a real testbed, and results show that ShuffleCAN can achieve better unbiased and efficient shuffling.

ACKNOWLEDGMENT

We sincerely thank the reviewers for their insightful feedback. This work was supported in part by the National Natural Science Foundation of China (#61972200, #61972199).

REFERENCES

[1] CAN Bus Can Be Encrypted, Says Trillium. https://www.eetimes.com/can-bus-can-be-encrypted-says-trillium. Accessed: 2015-10-22.

[2] CANcrypt - Scalable CAN Security. https://www.esacademy.com/en/products/cancrypt-details.html. Accessed: 2022-01-28.

[3] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck lightweight block ciphers. In *Proceedings of the 52nd annual design automation conference*, pages 1–6, 2015.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *International workshop on cryptographic hardware and embedded systems*, pages 450–466. Springer, 2007.

[5] M. Bozdal, M. Samie, and I. Jennions. A survey on CAN bus protocol: Attacks, challenges, and potential solutions. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pages 201–205. IEEE, 2018.

[6] R. Brown, A. Marti, C. Jenkins, and S. Shannigrahi. Dynamic Address Validation Array (DAVA) A Moving Target Defense Protocol for CAN bus. In *Proceedings of the 7th ACM Workshop on Moving Target Defense*, pages 11–19, 2020.

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[8] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, 2016.

[9] K.-T. Cho and K. G. Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1109–1123, 2017.

[10] G. J. Croll. Bientropy-the approximate entropy of a finite binary string. *arXiv preprint arXiv:1305.0954*, 2013.

[11] T. Dagan and A. Wool. Parrot, a software-only anti-spoofing defense system for the CAN bus. *ESCAR EUROPE*, page 34, 2016.

[12] R. A. Fisher, F. Yates, et al. *Statistical tables for biological, agricultural and medical research, edited by ra fisher and f. yates*. Edinburgh: Oliver and Boyd, 1963.

[13] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès. Lock it and still lose it—on the (In) Security of automotive remote keyless entry systems. In *25th USENIX security symposium(USENIX Security 16)*, 2016.

[14] B. Groza, L. Popa, and P.-S. Murvay. CANTO-Covert AutheNtication with Timing channels over Optimized traffic flows for CAN. *IEEE Transactions on Information Forensics and Security*, 16:601–616, 2020.

[15] K. Han, A. Weimerskirch, and K. G. Shin. A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier. *Proc. Eur. Embedded Secur. Cars (ESCAR)*, pages 13–29, 2015.

[16] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi. LSTM-Based Intrusion Detection System for In-Vehicle CAN Bus Communications. *IEEE Access*, 8:185489–185502, 2020.

[17] P. Kabal. Combinatorial coding and lexicographic ordering. 2018.

[18] C.-W. Lin and A. Sangiovanni-Vincentelli. Cyber-security for the controller area network (CAN) communication protocol. In *2012 International Conference on Cyber Security*, pages 1–7. IEEE, 2012.

[19] Z. Lu, Q. Wang, X. Chen, G. Qu, Y. Lyu, and Z. Liu. Leap: A lightweight encryption and authentication protocol for in-vehicle communications. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1158–1164, 2019.

[20] S. Nie, L. Liu, and Y. Du. Free-fall: Hacking tesla from wireless to CAN bus. *Briefing, Black Hat USA*, 25:1–16, 2017.

[21] B. Palaniswamy, S. Camtepe, E. Foo, and J. Pieprzyk. An efficient authentication scheme for intra-vehicular controller area network. *IEEE Transactions on Information Forensics and Security*, 15:3107–3122, 2020.

[22] A. Roy and S. K. Madria. CanSafe: An MTD based approach for providing resiliency against DoS attack within in-vehicle networks. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3243–3250. IEEE, 2022.

[23] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-allen and hamilton inc mclean va, 2001.

[24] Specification, CAN. Bosch. *Robert Bosch GmbH, Postfach*, 50, 1991.

[25] S. Woo, H. J. Jo, and D. H. Lee. A practical wireless attack on the connected car and security protocol for in-vehicle CAN. *IEEE Transactions on intelligent transportation systems*, 16(2):993–1006, 2014.

[26] S. Woo, D. Moon, T.-Y. Youn, Y. Lee, and Y. Kim. CAN ID shuffling technique (CIST): Moving target defense strategy for protecting in-vehicle CAN. *IEEE Access*, 7:15521–15536, 2019.

[27] T.-Y. Youn, Y. Lee, and S. Woo. Practical Sender Authentication Scheme for In-Vehicle CAN With Efficient Key Management. *IEEE Access*, 8:86836–86849, 2020.