Automotive Experiences
Vol. 9 No. 2 (2026) pp. 507-524
p-ISSN: 2615-6202 e-ISSN: 2615-6636

Research Paper

Enhancing CAN Bus Security via Lightweight Hardware-Based Identifier Randomization

Mohammed Saad Darouiche, El Bachir Tazi

Engineering Sciences Laboratory, Sidi Mohamed Ben Abdellah University, Fez, Morocco

mohammedsaad.darouiche@usmba.ac.ma

https://doi.org/10.31603/ae.15497

Published by Automotive Laboratory of Universitas Muhammadiyah Magelang

Article Info
Submitted:
03/12/2025
Revised:
11/07/2026
Accepted:
13/07/2026
Online first:
19/07/2026

Abstract

The controller area network (CAN) communication protocol used in vehicles relies on fixed message identifiers, which makes it vulnerable against frame injection and replay attacks. This study proposes an efficient lightweight hardware method that randomizes the identifier while preserving the priority rules that control bus arbitration. The design is implemented in a hardware description language (Verilog) and uses a linear feedback shift register (LFSR) as the randomization engine. The upper four bits of the identifier are kept unchanged to retain priority, where the lower seven bits are randomized. The module supports reseeding from a cryptographically secure random source. However, for the baseline statistical evaluation, reseeding was intentionally disabled to measure the intrinsic distribution. The design was evaluated using Xilinx Vivado environment. Statistical analysis was performed on 8,188 randomized ID, achieving a Shannon entropy of 6.999978 bits (maximum 7), and a chi-square goodness-of-fit test that showed no detectable deviation from a uniform distribution (χ^2 = 0.2482, p-value ≈ 1). Synthesis to a Artix-7 field-programmable device reported only 15 lookup tables and 23 flip-flops (<0.1% of resources), with a maximum operating frequency of 482 MHz, indicating a minimal hardware footprint. The mechanism was further validated on a physical CAN testbed confirming protection against replay and spoofing attempts, while the mechanism added no measurable bus or timing overhead. These results show that simple, hardware-level identifier randomization can strengthen in-vehicle communication while keeping arbitration behaviour intact and without requiring protocol changes.

Keywords: CAN bus; Identifier randomization; LFSR; Statistical analysis; Cybersecurity

1. Introduction

The CAN protocol, introduced by Bosch in the 1980s, is considered as the core standard for in-vehicle communication due to its efficiency, robustness, and low implementation cost. It enables multiple Electronic Control Units (ECUs) to exchange information reliably over a shared bus, reducing wiring complexity and ensuring real time data transmission [1]. However, despite its widespread adoption the CAN protocol was never designed with security in mind, where performance and determinism took over protection against malicious actors.

The CAN is a multimaster serial bus protocol originally developed to enhance communication between ECUs in vehicles. Unlike traditional point to point wiring systems, which significantly increased wiring complexity and vehicle weight, CAN allows all ECUs to communicate through a shared differential bus, thereby reducing wiring while maintaining high reliability and real time performance. Since its introduction, CAN has been standardized under ISO 11898, establishing its place as one of the most adopted communication protocols in the automotive industry [2].

CAN messages can be represented using **Figure 1**, where network nodes exchange data frames identified by either an 11-bit identifier in the standard frame format or a 29-bit identifier in

Figure 1. Standard CAN frame field

the extended frame format. The identifier is used to label the content of the message and ensure the priority of transmission within the network. During simultaneous transmissions, the arbitration mechanism guarantee that the message with the lowest identifier thus having the highest priority gains control of the bus without data loss or collisions. This deterministic behaviour makes CAN ideal for safety critical systems.

As **Figure 1** shows, the structure of the CAN frame places tight constraints on any security solution built on top of it, especially the identifier field that in addition of representing the message identity it holds the arbitration priority, so any scheme that alters the identifier has to keep the relative ordering of safety-critical messages intact. The payload adds a second limitation, since it carries at most 8 bytes (64 bytes in CAN-FD [3], which leaves very little space for appending authentication data. The most demanding constraint, however, is timing. Functions such as braking and steering rely on deterministic, real-time behaviour that limit the use of heavy cryptographic operations, as even a small amount of added latency can be enough to break the timing of a real-time control loop.

In recent years, numerous studies have demonstrated that the absence of built-in security mechanisms in CAN exposes vehicles to critical vulnerabilities such as spoofing, denial-of-service and replay attacks [4], [5], and with vehicles being increasingly connected via external interfaces such as Bluetooth, Wi-Fi and cellular networks, the attack surface of CAN expands significantly, raising urgent concerns about automotive cybersecurity [6]. As a result, several critical vulnerabilities have emerged, allowing attackers to use them to compromise vehicle functionality and safety [7]. One of the main vulnerabilities is that CAN frames are neither authenticated nor encrypted, thus any ECU connected to the bus and considered legitimate can inject messages into the network. This characteristic enables spoofing attacks, where an attacker transmits forged CAN frames that mimic authentic ECU messages to trigger unauthorized actions, such as disabling brakes or manipulating steering [7], [8]. Also, the fact that message integrity is not checked facilitates replay attacks, wherein valid messages captured from the network are retransmitted to elicit the same response from ECUs [9].

Another significant vulnerability is CAN's broadcast communication model, where every ECU receives all transmitted frames. This model enhances fault tolerance and synchronization but simultaneously exposes all message traffic to potential eavesdropping. Once an attacker gains physical or remote access to the bus, they can passively monitor and record traffic to infer sensitive information such as diagnostic codes or driver behaviour patterns [4]. This visibility also allows attackers to identify specific messages or ECUs for more targeted exploits.

The arbitration mechanism, designed to guarantee deterministic communication presents yet another exploitable feature. Because arbitration relies on the message identifier's numerical value to determine priority an attacker can monopolize bus access by transmitting frames with low IDs. This can result in a Denial of Service (DoS) condition, where legitimate ECUs are unable to transmit, disrupting critical operations such as engine control or braking [9], [10]. Although CAN employs error detection mechanisms such as CRC, bit monitoring and acknowledgment fields, these defences are used to protect against accidental faults rather than malicious behaviour. Attackers can use these mechanisms to create repeated error states forcing ECUs into a bus off state, effectively disconnecting them from the network [9], [10], [11].

To address these challenges, researchers have explored a variety of software and hardware-based approaches, ranging from Intrusion Detection Systems (IDSs) and cryptographic authentication schemes to identifier and message obfuscation methods [12], [13]. Among these

solutions, hardware-based approaches such as CAN identifier randomization are promising due to their low latency and compatibility with real-time requirements of embedded systems [14]. By introducing unpredictability at the identifier level, these techniques limit the predictability required for spoofing and replay attacks while maintaining deterministic communication essential for safety critical automotive applications [15].

Intrusion detection systems (IDS) represent the most widely studied defensive layer for in-vehicle networks. However, IDS-based approaches are inherently passive and reactive: rather than preventing attacks from happening, they monitor traffic and raise alerts after suspicious frames have already been transmitted to the bus. On the CAN network specifically, a message cannot be buffered and analyzed before continuing to its destination because latency strongly impact safety. Automotive IDS designers must simultaneously minimize both false positives and false negatives, a balance that is particularly difficult to achieve in safety-critical scenarios where a misclassified command (braking or steering) can directly trigger hazardous vehicle behaviour. Machine and deep learning-based approaches, while offering higher detection accuracy, are computationally intensive and impose resource requirements that conflict with the limited processing capacity of automotive ECUs [16].

Message authentication approaches like HMAC, AES-based MACs and standardised schemes such as AUTOSAR Secure Onboard Communication (SecOC) offer stronger security guarantees but face a distinct class of architectural constraints in the CAN domain. The frame payload is limited to 8 bytes, yet a truncated MAC typically requires 4–8 bytes of space alongside the actual message content. This mismatch requires multi-frame message splitting: AUTOSAR SecOC formalises this as an Authentic PDU and a separately transmitted Secured PDU [17], directly increasing frame count and bus load for every secured message. On the computational side, software-only AES-based MAC generation imposes latencies of up to 12 ms on representative automotive-grade microcontrollers [18] a value that violates the hard real-time deadlines of time-triggered CAN networks [19]. These limitations motivate a complementary approach that operates at the identifier level, below the software and protocol stacks, and introduces neither transmission latency nor payload overhead.

Based on that, this paper proposes a hardware-based CAN identifier randomization mechanism that enhances bus level security while preserving CAN arbitration. The approach is built upon pseudo random number generation (PRNG) techniques integrated at the hardware level, enabling lightweight yet effective obfuscation of CAN identifiers. Unlike purely software mitigations, this solution is designed for Field-Programmable Gate Array (FPGA) or ECU integration ensuring minimal computational overhead and compatibility with existing automotive architectures. The specific research gap addressed is the absence of a protocol-transparent, hardware-level randomization solution that: (i) operates below the software stack without modifying CAN controllers or ECU firmware, (ii) preserves bus arbitration by fixing the identifier MSBs, (iii) introduces zero additional latency in the CAN frame transmission path and (iv) evaluated for randomness, implemented and synthesized on an FPGA and validated against replay and spoofing attacks on a physical CAN testbed, a combination that, to the best of our knowledge is not commonly reported together in prior lightweight CAN identifier-randomization approaches.

2. Methods

2.1. Threat Model

The proposed mechanism is designed to defend against an adversary with physical or remote bus-level access for example, through the OBD-II diagnostic port, a compromised ECU or a wireless gateway who can passively observe CAN frames and actively inject arbitrary frames. This threat profile is consistent with demonstrated real-world vehicle attacks [4], [5], [6], [7], [8] and represents the primary attack surface of unprotected CAN networks. The adversary is assumed to have no direct physical access to the internals of legitimate ECUs and therefore cannot read the LFSR state register or the initialization seed from hardware.

Three specific attack objectives are considered in this work: (1) Replay attacks, in which the adversary records a legitimate CAN frame and retransmits it at a later time to trigger an

unauthorized actuation, (2) Spoofing attacks, in which the adversary reuses the identifier of a legitimate ECU together with a crafted payload, in order to impersonate that ECU and inject falsified data or commands and (3) Identifier prediction attacks, in which the adversary observes a sequence of randomized identifiers and attempts to compute future valid identifiers without direct knowledge of the LFSR state.

2.2. *Simulation and Data Collection*

The proposed Secure_CAN_ID_Randomizer module was validated through a series of simulations using Xilinx Vivado. The simulation environment was designed with the following components:

a. Clock and Reset Generator: A 100 MHz system clock was generated, with synchronous reset asserted during initialization.
b. Original CAN ID Input: A fixed 11-bit identifier was injected into the module to represent a standard CAN frame source.
c. Seed Update Logic: The update_seed signal is supported and was exercised in separate functional runs to verify reseeding behaviour, but for the baseline static dataset used for statistics, reseeding was intentionally disabled after the initial seed load.

The randomized CAN ID output and top-level control signals (clk, reset, update_seed) were monitored in the Vivado waveform viewer to verify correct operation (**Figure 2**).

Each randomized CAN ID was exported to a CSV file at every update event using $fopen, $fwrite, and $fclose system tasks in Verilog. This CSV file represents the raw dataset organized to include the simulation cycle index and the generated randomized CAN IDs.

2.3. *Architecture of the CAN ID Randomizer*

This subsection presents the structural design of the randomizer module implemented in Verilog. It is composed of three core logic blocks: a PRNG based on LFSR, a seed update mechanism, and the logic for merging randomized bits with the original CAN ID. The developed Verilog module, named Secure_CAN_ID_Randomizer is shown in **Figure 3**.

2.4. *Pseudo Random Number Generator Module*

A PRNG is a deterministic algorithm that produces a series of numbers that mimics the statistical properties of random numbers. Unlike true randomness, which may be derived from physical processes (thermal noise, quantum events), PRNGs are algorithmic and thus fully determined by an initial state or seed [20]. Formally, a PRNG is a deterministic function expressed in Eq. (1).

$$X_{n+1} = f(X_n) \tag{1}$$

where X_n represents the current state, and f is the recurrence function that computes the next state. The generated sequence $(X_0, X_1, X_2, \ldots)$ is determined by the seed X_0 .

Several classes of PRNGs exist, each with different characteristics in terms of computational complexity, randomness quality and suitability

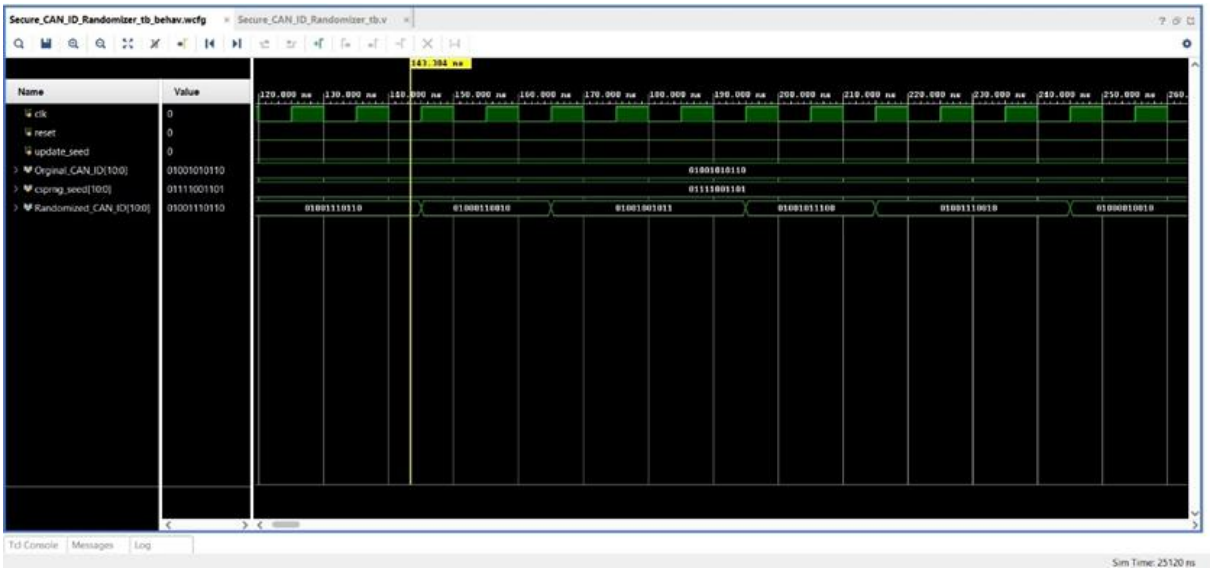


Figure 2. Signals observation under Vivado waveform viewer

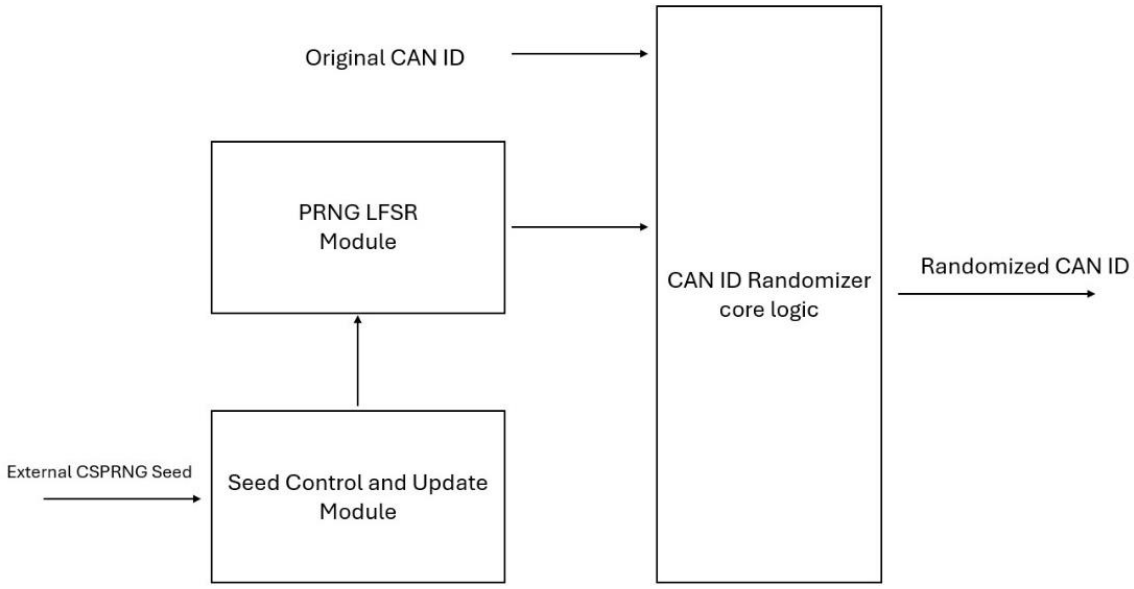


Figure 3. Block diagram of the Secure CAN ID Randomizer architecture

for hardware. Given the strict real time and hardware constraints of automotive systems, Linear Feedback Shift Register-based PRNGs offer an attractive compromise between lightweight implementation and statistical quality [21].

An LFSR is a shift register where the new input bit is the XOR of selected "tap" bits from the register. An n bit LFSR can be defined using Eq. (2):

$$S_{n+1} = (S_n \ll 1) \oplus f(S_n[taps]) \quad (2)$$

Where S_n is the current state at step n, S_{n+1} is the next state after the shift operation and $taps$ are specific bit positions chosen to maximize period length. The LFSR can be built using only XOR gates and flip-flops, producing sequences with periods up to $2^n - 1$. This generator is widely used in communication systems, but its linear nature makes it predictable if enough output bits are known, so it's not cryptographically secure when used alone. However, their unpredictability can be enhanced by periodically reseeding them with values generated from a stronger source, such as Cryptographically Secure Pseudorandom Number Generator (CSPRNG). Our LFSR is composed of an 11 bits shift register that updates its state at every clock cycle based on a set of feedback taps. The taps define which register bits are XORed together to compute the new input bit. Mathematically, the next state of the LFSR can be described using Eq. (3):

$$S_{n+1} = ((s_{n-1}, s_{n-2}, \dots, s_1), f(s_{10}, s_8, s_6, s_5)) \quad (3)$$

where, the XORed taps are expressed using Eq. (4).

$$f(s_{10}, s_8, s_6, s_5) = s_{10} \oplus s_8 \oplus s_6 \oplus s_5 \quad (4)$$

This configuration ensures a maximal length sequence of $2^{11} - 1 = 2047$ states before repetition, which is sufficient for CAN ID randomization purposes. In the dataset reported in this paper, we used a primitive 11 bit polynomial with period 2047 and we did not reseed while collecting the randomized identifiers and this let the LFSR run through complete periods, so the uniformity results reflect only its inherent distribution, not artifacts from mid-cycle resets. The update mechanism of the LFSR is described in Algorithm 1, where the next state S_{n+1} is derived from the current state S_n by applying a XOR operation on selected tap positions, followed by a left shift.

Algorithm 1: Fibonacci LFSR State Update

Input: Current state $S_n = [s_{10}, s_9, \dots, s_0]$
Output: Updated state S_{n+1}
Begin:
- **Compute** the feedback bit: $f = s_{10} \oplus s_8 \oplus s_6 \oplus s_5$
- **Shift** all bits left
- **Insert** feedback bit at position s_0
- **Return** the updated state S_{n+1}

End:

The choice of taps is crucial because poorly chosen taps would reduce the sequence length and impact negatively the randomness.

2.5. Seed Control and Update Module

One limitation of basic LFSRs is that once initialized, their sequence is deterministic and as a result an attacker who observes enough outputs can potentially reconstruct the state and predict future values. To mitigate this risk, we added external seed update functionality that simulate the reseeding mechanism generated by a CSPRNG as shown in Algorithm 2.

Algorithm 2: LFSR Reseeding

Input: Current LFSR state S_n, external seed *csprng_seed*, *Reset* and *Update_seed*
Output: Updated state S_{n+1}
Begin:

- **If** *Reset* = 1 **then**
  $S_{n+1} = csprng\_seed$ (Default value)
- **Else if** *Update_seed* = 1 **then**
  $S_{n+1} = csprng\_seed$
- **Else**
  Update the state using Algorithm 1

End:

2.6. CAN ID Randomizer Core Logic Module

The CAN arbitration mechanism relies strongly on the identifier's MSBs to resolve priority and this is why the randomizer module keeps the 4 MSBs unchanged, while randomizing the 7 LSBs. The recomposed CAN ID can be defined using Eq. (5) as:

$$CAN\_ID_{rand} = (CAN_{ID\,orig}[10:7], S_n[6:0]) \quad (5)$$

where, CAN_ID_{rand} is the randomized CAN ID, CAN_ID_{orig} the original CAN ID, and $S_n[6:0]$ are the 7 least significant bits of the LFSR output. This ensures that arbitration is preserved, while introducing entropy and unpredictability into the lower bits, thus making spoofing and replay significantly harder.

A key operational requirement of identifier randomization is that authorized receiver ECUs must be able to determine which identifiers are currently valid. In the proposed architecture, each authorized receiver maintains a local copy of the LFSR, initialized with the same seed value and primitive polynomial as the transmitter through a secure provisioning channel consistent with standard automotive key provisioning practices. At each frame interval, the receiver advances its local LFSR by one step and computes the expected identifier using the same bijective mapping defined in Eq. (5). A received frame is accepted if and only if its identifier matches the locally expected value, any mismatch is treated as an anomalous or potentially malicious frame and rejected accordingly.

Because the CAN data-link layer already provides CRC validation, acknowledgment and bit-level error detection, any frame corrupted by a transient bus error or delayed by arbitration contention is detected by every node and automatically retransmitted by the sender with the same identifier. Since the receiver advances its LFSR only on frames that the CAN controller has validated (correct CRC and acknowledgment), such events do not cause loss of synchronization: the transmitter and receiver each advance their LFSR exactly once per successfully delivered message. This behaviour was confirmed in the physical testbed, where the receiver remained synchronized with the transmitter throughout operation and correctly rejected all injected replay and spoofing frames.

2.7. Post Processing

At the end of the simulation the raw dataset was generated containing 8,188 samples of randomized CAN identifiers that represent exactly four full LFSR periods, the purpose was to avoid partial period sampling artifacts. This dataset was not yet ready for analysis, so first we convert each randomized CAN ID originally stored as an 11 bits binary string into decimal format for easier manipulation in Python, then we only extracted the 7 randomized LSBs since the 4 MSBs remained unchanged.

2.8. Statistical Analysis

Once the dataset is prepared, a rigorous statistical analysis was performed to evaluate the randomness, unpredictability and uniformity of the proposed design. This step was crucial to verify statistical randomness quality. The analysis was done using two complementary techniques, Shannon Entropy to measure the unpredictability of the sequence and Chi-Square Goodness-of-Fit test to evaluate the identifiers uniformity.

Shannon Entropy measurement - Entropy quantifies the level of unpredictability within a dataset and is determined by computing the average amount of information each data item contains [22]. Shannon entropy has been widely

applied in anomaly detection and IDS for CAN traffic, where changes in identifier entropy can reveal malicious activity. Also, it is considered as a standard metric for evaluating randomness quality in secure systems [23]. In our study entropy is used to quantify the unpredictability of the generated identifiers.

For a discrete random variable X taking values $x_1, x_2, \dots, x_n$ with probability function $p(x_i)$, entropy is defined as Eq. (6):

$$H(X) = -\sum_{i=1}^{n} p(x_i).\log_2(p(x_i)) \quad (6)$$

Where $p(x_i)$ is the probability of occurrence of the i^{th} possible value x_i.

A value of $H(X) = 0$ corresponds to a completely predictable outcome with no randomness, where on the other hand a value of $H(X) = \log_2(n)$ corresponds to a perfectly uniform distribution over n values. In our case, only the 7 LSBs are randomized so the maximum entropy will be expressed using Eq. (7).

$$H_{max} = \log_2(2^7) = 7 \quad (7)$$

Thus, entropy values approaching 7 indicate that the randomizer achieves near perfect unpredictability, while lower values would suggest bias or predictability.

Chi-Square Uniformity Test - This test is a fundamental statistical tool for assessing the randomness quality of generated sequences and determining how well observed frequencies align with expected probabilities under a uniform distribution hypothesis. The Chi-Square test is widely recognized in the evaluation of PRNGs and forms a core component of the "NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (SP 800-22)" [24]. It has been extensively used in the validation of hardware and software based PRNGs, particularly within embedded and cryptographic systems [25]. In this study, the Chi-Square test was applied to the sequence of randomized CAN identifiers to verify whether their frequency distribution across possible identifier values deviated significantly from the ideal uniform model. This test compares the observed frequency of each possible outcome with the expected frequency under the null hypothesis of uniformity.

We've defined the Null Hypothesis H_0 as the CAN IDs uniformly distributed, which means each possible ID has the same probability of appearing. On the other hand, the Alternative Hypothesis H_a was defined as the CAN IDs being not uniformly distributed, where some ID occur more frequently than others. Eq. (8) represents the Chi-Square test statistics:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \quad (8)$$

Where O_i is the observed frequency of the outcome i, and E_i is the expected frequency under uniform distribution given by Eq. (9).

$$E_i = \frac{N}{k} \quad (9)$$

With N representing the total number of samples collected, and k the number of categories. We also compute the Pearson residuals (Eq. 10) and per-bin Chi-square contributions (Eq. 11), that will be used later in the Results and Discussion section.

$$r_i = \frac{(O_i - E)}{\sqrt{E}} \quad (10)$$

Where r_i is the Pearson residual for each category i.

$$c_i = \frac{(O_i - E)^2}{E} \quad (11)$$

Where c_i is the per-bin chi-square contribution. The test follows a Chi-Square distribution with degrees of freedom $df = k - 1$ Instead of comparing the statistic to a critical value from Chi-Square tables, we used the p-value approach, so if $p < 0.05$ we reject the Null Hypothesis H_0 with a conclusion that the data is not uniform, but if $p > 0.05$ then we failed to reject the Null Hypothesis H_0 saying that the data has a uniform distribution.

2.9. Hardware Synthesis

To validate the hardware feasibility of the proposed design, the Secure_CAN_ID_Randomizer module was synthesized using Xilinx Vivado 2024.2, targeting a Artix-7 FPGA (XC7A35T-1CPG236C) device. **Figure 4** provides a clear structural overview of the implemented logic that shows the use of lightweight combinational logic (XOR gates and multiplexers) and a minimal number of sequential elements (flip-flops). These features ensure that the design achieves low resource utilization, thus a minimal propagation delay needed by real-time automotive communication systems.

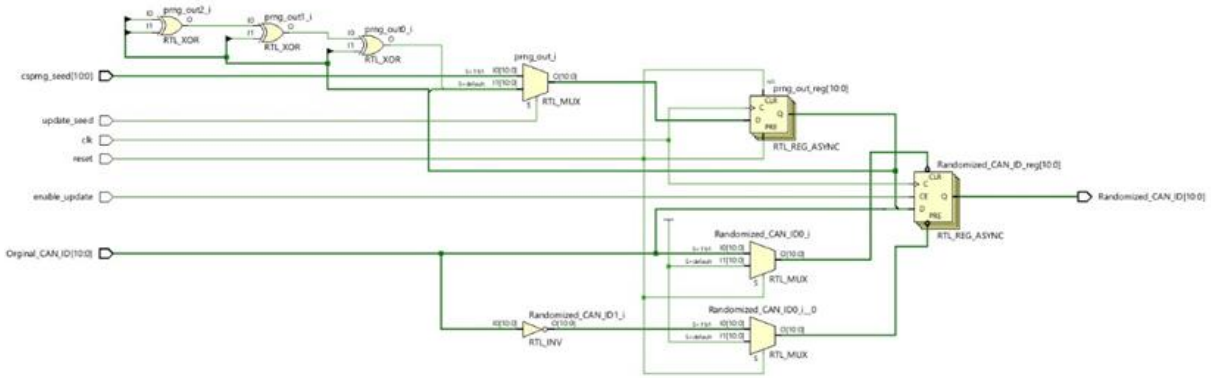


Figure 4. RTL schematic of the Secure_CAN_ID_Randomizer module

3. Results and Discussion

This section presents the experimental results obtained from the simulation of the proposed CAN ID Randomizer, the statistical evaluation of randomness and the hardware synthesis analysis.

3.1. Simulation Results

The Verilog module was simulated in Vivado using a dedicated testbench to generate 8,188 randomized CAN identifiers. Reseeding was intentionally disabled after the initial non-zero seed load. **Figure 5** shows the simulation waveform of the Secure_CAN_ID_Randomizer module in Vivado where we can visualize the Clk driving the system with a stable periodic signal, Reset signal which is initially activated, after which the module begins normal operation, the Original CAN ID fixed to 0x256, which corresponds to the identifier before randomization, the csprng_seed input set to 0x5AB (supported for reseeding in functional tests) and the Randomized_CAN_ID output that shows a sequence of different values (0x261, 0x204, 0x225, 0x257, 0x238, etc.) confirming that the 7 LSBs of the CAN identifier are continuously updated according to the LFSR evolution.

The simulation waveform shown in **Figure 5** confirmed correct operation of the LFSR feedback mechanism, with continuous pseudo-random bit generation. We can see also that our design successfully preserved the 4 MSBs of the original CAN ID needed for conserving arbitration set before the randomization process, while the 7 randomized LSBs varied across the simulation.

3.2. Statistical Evaluation of Randomness

To perform the statistical analysis two Python scripts were developed to execute the required computations on the generated CSV dataset.

Shannon Entropy Analysis - **Figure 6** presents the probability distribution of the randomized CAN identifiers obtained from a dataset of 8,188

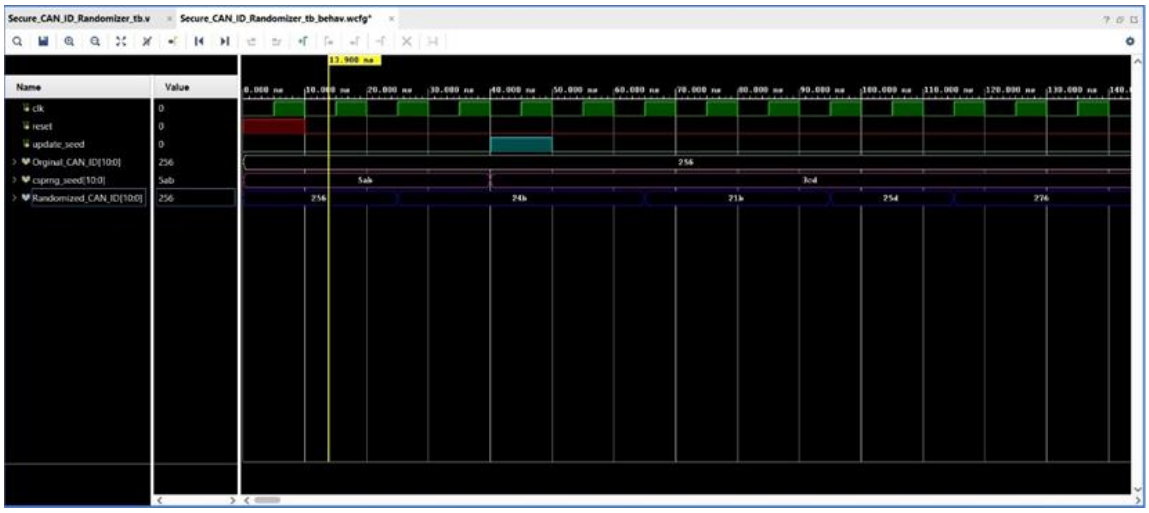


Figure 5. Simulation waveform of the Secure_CAN_ID_Randomizer module in Vivado

samples. Each bar represents the probability of occurrence of a specific identifier value after randomization. The histogram shows an evenly spread pattern across the entire identifier range within the preserved MSB class, with no visible clustering or dominant peaks. This visual uniformity indicates that the generated identifiers are well-distributed and do not favor specific numerical regions, which is a desirable property for randomization mechanisms in automotive networks. The computed Shannon entropy value is H = 6.999978 bits (maximum 7). The absence of repetitive or biased patterns further supports the effectiveness of the implemented PRNG in generating statistically well-dispersed outputs.

Chi-Square Uniformity Test - To assess uniformity, we applied Chi-square goodness-of-fit test to the 7-bit randomized field (k = 128 categories) under uninterrupted full-period operation (N = 8,188 samples representing four LFSR periods). The expected count per category under H_0 was $E = 63.97$ and $\chi^2 = 0.2482$ with $df = 127$ and p-value ≈ 1, with those results we failed to reject H_0, indicating no evidence of deviation from a uniform distribution.

To visualize where departures occur, we use a combined Pearson residual / Chi-square contribution plot (**Figure 7**), using r_i and c_i as defined in the Method section. All residuals lie within ±0.50 (reference bands at ±2 and ±3 is not approached). This pattern confirms the distribution is as balanced as mathematically possible for the chosen sample size and period structure. The single under-represented bin arises from the indivisible period of the LFSR which forces counts of 60 for one category and 64 for the remaining 127 over four periods.

To directly address whether the periodic reseeding mechanism described in Algorithm 2 affects statistical uniformity, a dedicated simulation compared two operating modes over the same sample count (N = 8,188 frames): (i) continuous operation without reseeding across four full LFSR periods (the baseline configuration) and (ii) operation with periodic reseeding every P = 3 frames, each new seed drawn from a simulated

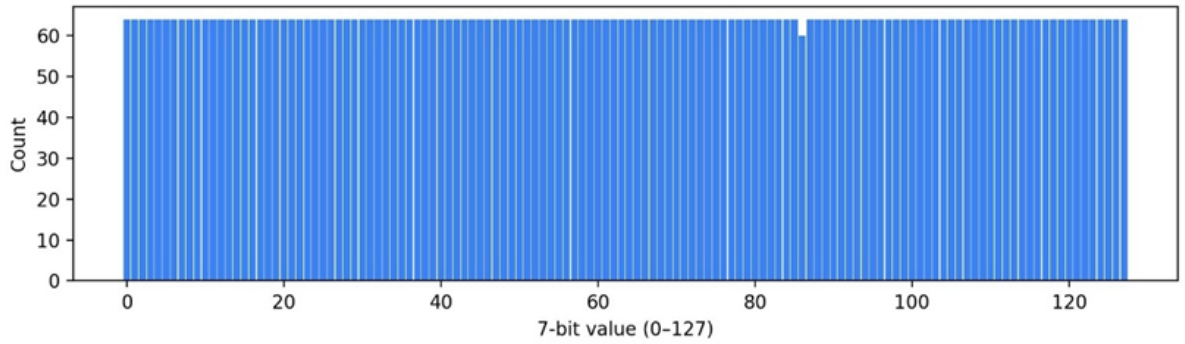


Figure 6. Probability distribution of the randomized CAN identifiers

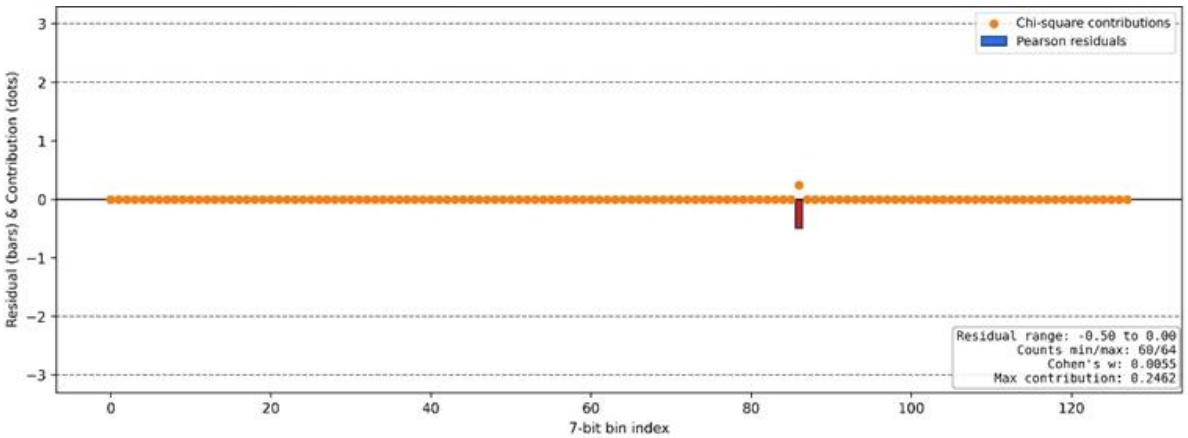


Figure 7. Combined Pearson residual and chi-square contribution plot for the 7-bit randomized identifier distribution

CSPRNG. The value P = 3 was chosen based on the security analysis in Section 3.3, so no attacker can gather enough consecutive bits between two reseeding operation needed for ID reconstruction.

As shown in **Figure 8**, periodic reseeding reduces Shannon entropy from H = 6.999978 bits (no reseeding) to H = 6.990876 bits (P = 3), a reduction of 9.1×10^{-3} bits less than 0.13% of the theoretical 7-bit maximum. The chi-square statistic under reseeding ($\chi^2 = 103.96$, df = 127, p = 0.933) remains well within the acceptable range for uniformity. Under the no-reseeding baseline, the near-zero chi-square ($\chi^2 = 0.2482$, p = 1) reflects the fully deterministic structure of a single-seed LFSR cycle, where 127 bins receive exactly 64 counts and one bin receives exactly 60 counts over four periods. Reseeding replaces this deterministic pattern with independent, randomly seeded segments, producing realistic chi-square values while preserving near-maximum entropy. These results confirm that periodic reseeding strengthens security against state reconstruction attacks without measurably degrading the distributional uniformity of the randomized identifiers.

3.3. *Security Analysis*

The LFSR is a linear, deterministic generator and is therefore not cryptographically secure on its own. The statistical uniformity established in Section 3.2 confirms that the randomized identifiers are well distributed, but a uniform distribution does not, by itself, guarantee that the sequence cannot be predicted. This section examines how resistant the scheme is to an adversary who attempts to reconstruct the LFSR state from observed bus traffic.

Due to this linearity, an attacker can apply the Berlekamp–Massey algorithm [26], which recovers the complete state of an L-bit maximal-length LFSR from roughly 2L output bits [27] (in our case (in our case 2 × 11 = 22)). Each transmitted frame reveals seven of these bits (the randomized part of the identifier), an attacker therefore needs only about four consecutive frames ($\lceil 22/7 \rceil \approx 4$) to reconstruct the state and from then on, predict every identifier the LFSR will generate.

This is the fundamental limitation of plain LFSR randomization, and it is precisely what the periodic reseeding mechanism of Section 2.5 (Algorithm 2) addresses. If the LFSR is reseeded from a CSPRNG more frequently than this reconstruction window (a period of two to three frames is recommended) the attacker's observations are reset before enough bits accumulate, and Berlekamp–Massey reconstruction becomes infeasible.

3.4. *Hardware Synthesis Results*

To validate hardware feasibility, the proposed Secure_CAN_ID_Randomizer module was synthesized and implemented using Xilinx Vivado 2024.2, targeting the Artix-7 FPGA (XC7A35T-1CPG236C) (the same device used for physical

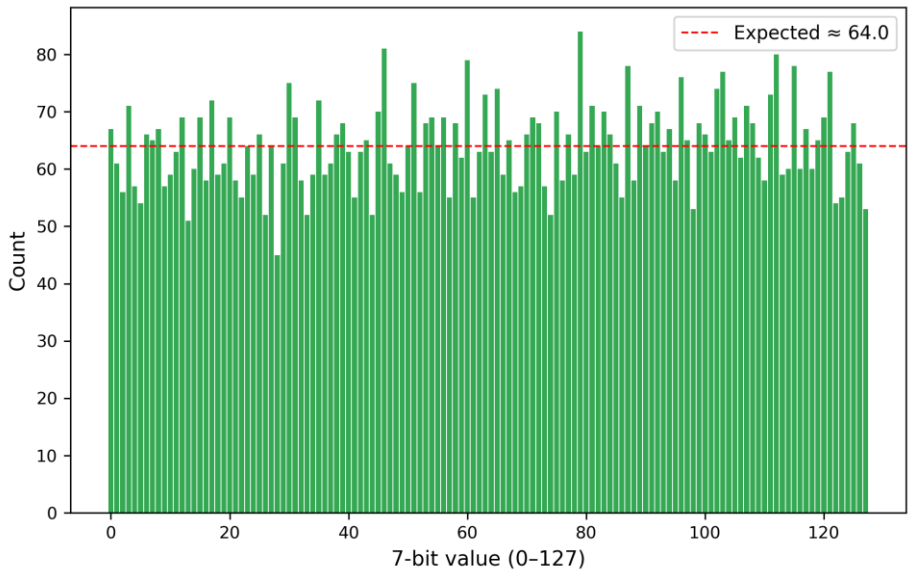


Figure 8. Distribution of the 7-bit randomized identifier field under periodic reseeding (P = 3 frames, N = 8188 frames)

hardware validation in this work). The RTL schematic, shown in Figure 4, demonstrates the use of XOR gates, multiplexers and flip-flops, with no DSP blocks or block RAMs required.

As reported in Table 1, the design consumes only 15 Slice LUTs and 23 flip-flops, representing 0.07% and 0.06% of the available Artix-7 resources respectively. Synthesis completed with zero errors and zero critical warnings. Timing analysis under a 100 MHz clock constraint yields a worst-case critical path delay of 2.072 ns, corresponding to a maximum operating frequency (Fmax) of 482 MHz and a Worst Negative Slack (WNS) of 7.928 ns and no failing endpoints, the design meets the required timing constraints, which means the randomizer adds no delay to the CAN bus clock domain and would still work comfortably at CAN-FD data rates of up to 8 Mbit/s.

Power analysis reports a total on-chip power of 76 mW, of which 92% (70 mW) is FPGA device static power independent of the design logic. The dynamic power directly attributable to the randomizer logic is less than 1 mW. In an integrated ECU or ASIC implementation, where I/O buffers are shared with the existing CAN controller, the effective power overhead of the proposed module is negligible compared to the typical automotive ECU power budget of 200–500 mW. The junction temperature under nominal conditions is 25.4 °C, with a thermal margin of 59.6 °C, confirming safe operation across the full automotive temperature range.

3.5. *Validation under Hardware Testbed*

To complement the theoretical and simulation-based validation, a three-node physical CAN bus testbed was used, as illustrated in Figure 9. The transmitter node is implemented on a Basys-3 FPGA board equipped with an SN65HVD230 CAN transceiver. This node executes the Fibonacci LFSR randomizer entirely in hardware at 100 MHz and transmits one CAN 2.0A standard frame per second, using a rolling 8-bit counter as payload. The original CAN identifier is set to 0x123 via on-board slide switches, when the randomizer is activated (SW[15] = 1), the lower 7 bits are XOR-masked with the LFSR keystream, producing a different randomized identifier each period while preserving the upper 4 priority bits. The legitimate receiver is a Raspberry Pi Pico programmed in MicroPython connected to the bus via an MCP2515 SPI CAN controller and a TJA1050 transceiver. It runs a software mirror of the same LFSR and filters incoming frames against the expected randomized identifier. The attacker node is an Arduino Uno with an identical MCP2515 module. It first passively sniffs the bus to capture the target identifier, then launches replay or spoofing attacks at 200 ms intervals via serial commands. All three nodes share a two-wire CAN bus operating at 500 kbps with 8 MHz crystal oscillators on the MCP2515 modules.

Replay attack scenario - The replay attack scenario follows three phases. In the first phase, the attacker node passively monitors the bus in sniff mode and captures legitimate FPGA frames (ID = 0x123, incrementing payload bytes 0x32 to 0x6E), as shown in Figure 10. In the second phase, the attacker activates the replay loop and continuously retransmits the last captured frame at 200 ms intervals, as shown in Figure 11.

Figure 12 shows the legitimate receiver response in the unprotected configurations (ID randomizer deactivated). In this mode the receiver accepts every received frame regardless

Table 1. Resource utilization and performance summary for Xilinx Artix-7 (XC7A35T-1CPG236C)

Resource Type	Used	Available	Utilization (%)
Slice LUTs	15	20,800	0.07
Flip-flops	23	41,600	0.06
Block RAMs	0	50	0.00
DSP Slices	0	90	0.00
Bonded IOBs	38	106	35.85
BUFG (clocks)	1	32	3.13
Fmax	482 MHz	—	—
Critical path delay	2.072 ns	—	—
WNS	7.928 ns	—	—
Core logic power	<1 mW	—	—
Total on-chip power	76 mW	—	—

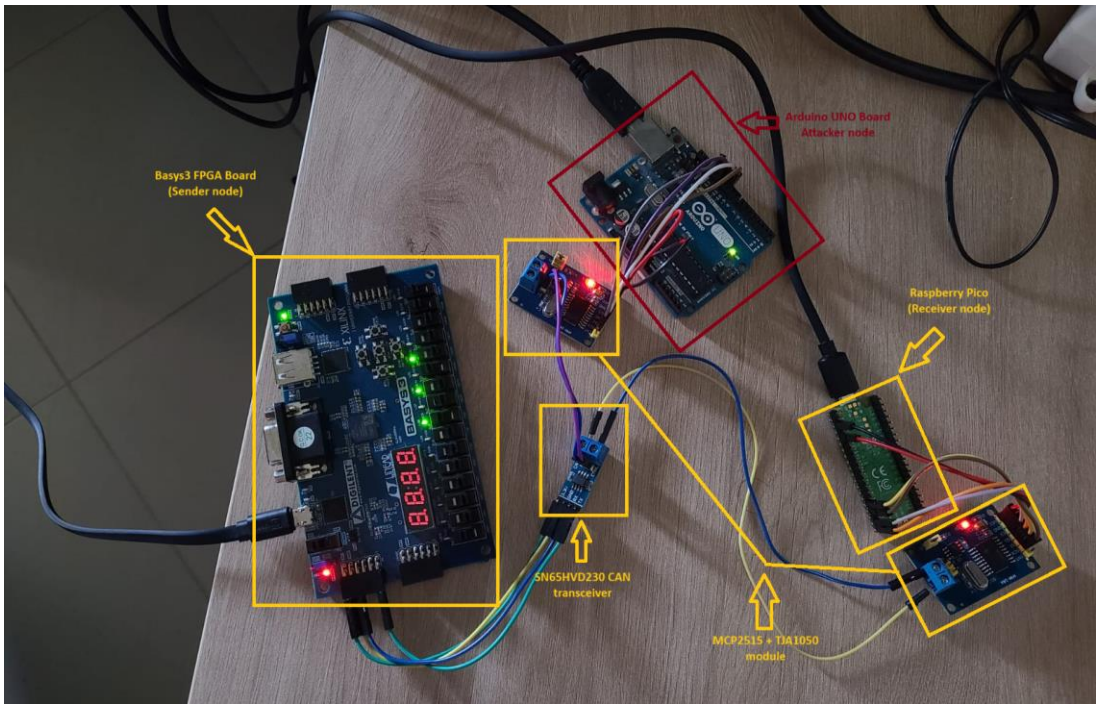


Figure 9. Physical CAN testbed.

```
Output   Serial Monitor ×
Message (Enter to send message to 'Arduino UNO' on 'COM6')

=== Arduino CAN Attacker ===
Initialising MCP2515 at 500 kbps (8 MHz crystal)...
Entering Configuration Mode Successful!
Setting Baudrate Successful!
MCP2515 ready.
Commands: r=replay once  a=attack loop  s=stop  p=show captured
------------------------------------------------------------

[ SNIFF MODE ]
[SNIFF]  ID=0x123  DLC=1  Data=[32]
[SNIFF]  ID=0x123  DLC=1  Data=[3A]
[SNIFF]  ID=0x123  DLC=1  Data=[56]
[SNIFF]  ID=0x123  DLC=1  Data=[5E]
[SNIFF]  ID=0x123  DLC=1  Data=[6E]
```

Figure 10. Attacker captures FPGA frames (sniffing mode)

```
[ SNIFF MODE ]
[SNIFF]  ID=0x123  DLC=1  Data=[32]
[SNIFF]  ID=0x123  DLC=1  Data=[3A]
[SNIFF]  ID=0x123  DLC=1  Data=[56]
[SNIFF]  ID=0x123  DLC=1  Data=[5E]
[SNIFF]  ID=0x123  DLC=1  Data=[6E]
[ ATTACK LOOP ] Replaying every 200 ms. Press 's' to stop.
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
[TX REPLAY]  ID=0x123  DLC=1  Data=[6E]
```

Figure 11. Replay attack

```
Shell
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Legitimate receiver ready — PLAIN (LFSR_Randomizer not activated)
Original ID = 0x123
-------------------------------------------------------------
    #      ID  DLC  Data                    Status
-------------------------------------------------------------
    1  0x123    1  [32                  ]  OK
    2  0x123    1  [3A                  ]  OK
    3  0x123    1  [56                  ]  OK
    4  0x123    1  [5E                  ]  OK
    5  0x123    1  [6E                  ]  OK
    6  0x123    1  [6E                  ]  OK
    7  0x123    1  [6E                  ]  OK
    8  0x123    1  [6E                  ]  OK
    9  0x123    1  [6E                  ]  OK
   10  0x123    1  [6E                  ]  OK
   11  0x123    1  [6E                  ]  OK
   12  0x123    1  [6E                  ]  OK
   13  0x123    1  [6E                  ]  OK
   14  0x123    1  [6E                  ]  OK
   15  0x123    1  [6E                  ]  OK
   16  0x123    1  [6E                  ]  OK
   17  0x123    1  [6E                  ]  OK
   18  0x123    1  [6E                  ]  OK
   19  0x123    1  [6E                  ]  OK
   20  0x123    1  [6E                  ]  OK
   21  0x123    1  [6E                  ]  OK
   22  0x123    1  [6E                  ]  OK
   23  0x123    1  [6E                  ]  OK
   24  0x123    1  [6E                  ]  OK
```

Figure 12. Receiver node in non-protected mode

of whether it originated from the FPGA or the attacker. Frames 1 to 5 correspond to legitimate FPGA transmissions, while frames 6 to 24 are attacker replays all are logged as OK, confirming the vulnerability of static CAN identifiers to replay attacks. When the LFSR randomizer is activated, the receiver maintains a local LFSR state initialized from the shared seed and computes the expected randomized identifier before every reception. All 20 replay frames carrying the static identifier 0x123 are immediately rejected with the message "ATTACK expected 0x121, REJECTED" as shown in **Figure 13**, because the expected identifier advances every frame, so a previously captured one is already stale and outdated by the time it is replayed and can no longer match.

Spoofing Attack Evaluation - After identifying the target identifier (0x123) during the sniffing phase, the attacker node activates the spoof loop and transmits forged frames carrying a malicious payload (0xFF) at 200 ms intervals.

Figure 14 presents the receiver behaviour under the spoofing attack in the non-protected mode, where the legitimate receiver accepts all incoming frames with ID = 0x123 indistinguishably from authentic FPGA traffic. The injected payload 0xFF is logged as OK across all 17 received frames, confirming a successful spoofing attack. When the LFSR randomizer is activated, the receiver rejects all 20 spoofed frames with the message "ATTACK expected 0x121, REJECTED" as shown in **Figure 15**. Since the attacker has no knowledge of the current LFSR state and cannot reproduce the correct randomized identifier, so the attack is blocked regardless of the injected payload.

Timing and Bus-Load Analysis - A critical requirement for any in-vehicle security mechanism is that it must not degrade the deterministic timing of the CAN bus. The proposed randomizer computes the randomized identifier in a single clock cycle, so at 100 MHz operating frequency, this corresponds to a latency of 10 ns, while the underlying logic settles in only 2.072 ns (Section 3.4). Because the randomized identifier is produced during the inter-frame space before the SOF bit is placed on the bus the operation is fully hidden within the existing transmission pipeline and introduces no delay into the frame itself.

In addition to that, the mechanism preserves the CAN bus load by construction, because randomization only remaps the value of the 7 identifier's LSB through a XOR operation and does not introduce any additional handshake or key-exchange frames, also does not alter the frame format, the data length code or the transmission period. Consequently, both the number of bits per frame and the number of frames per second remain identical to those of a non-randomized CAN network, so the resulting bus utilization is unchanged. This is a fundamental advantage over message-authentication schemes (e.g., HMAC- or AES-MAC-based approaches), which transmit additional authentication data and therefore increase bus load and latency.

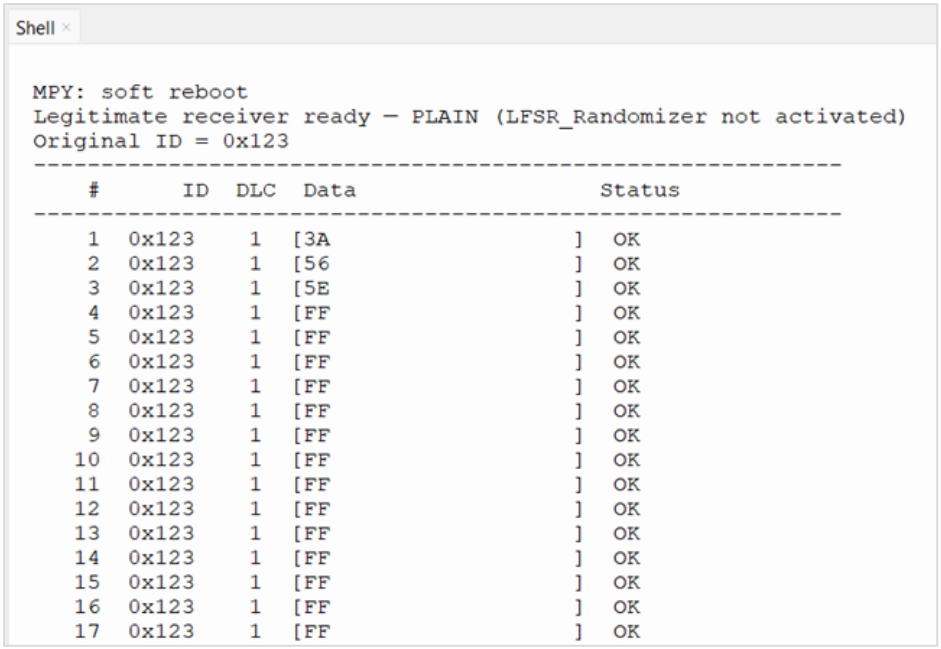
3.6. Comparative Discussion

In the context of automotive embedded security, several recent contributions provide relevant points of comparison to our proposed identifier randomization mechanism.

For instance, Han *et al.* [28] introduced the Identity-Anonymized CAN (IA-CAN) protocol, which randomizes CAN identifiers to provide

```
Original ID = 0x123
------------------------------------------------------------
    #      ID  DLC  Data                  Status
------------------------------------------------------------
    1  0x123    1  [5E                ]  ** ATTACK #1 – expected 0x121, REJECTED **
    2  0x123    1  [5E                ]  ** ATTACK #2 – expected 0x121, REJECTED **
    3  0x123    1  [5E                ]  ** ATTACK #3 – expected 0x121, REJECTED **
    4  0x123    1  [5E                ]  ** ATTACK #4 – expected 0x121, REJECTED **
    5  0x123    1  [5E                ]  ** ATTACK #5 – expected 0x121, REJECTED **
    6  0x123    1  [5E                ]  ** ATTACK #6 – expected 0x121, REJECTED **
    7  0x123    1  [5E                ]  ** ATTACK #7 – expected 0x121, REJECTED **
    8  0x123    1  [5E                ]  ** ATTACK #8 – expected 0x121, REJECTED **
    9  0x123    1  [5E                ]  ** ATTACK #9 – expected 0x121, REJECTED **
   10  0x123    1  [5E                ]  ** ATTACK #10 – expected 0x121, REJECTED **
   11  0x123    1  [5E                ]  ** ATTACK #11 – expected 0x121, REJECTED **
   12  0x123    1  [5E                ]  ** ATTACK #12 – expected 0x121, REJECTED **
   13  0x123    1  [5E                ]  ** ATTACK #13 – expected 0x121, REJECTED **
   14  0x123    1  [5E                ]  ** ATTACK #14 – expected 0x121, REJECTED **
   15  0x123    1  [5E                ]  ** ATTACK #15 – expected 0x121, REJECTED **
   16  0x123    1  [5E                ]  ** ATTACK #16 – expected 0x121, REJECTED **
   17  0x123    1  [5E                ]  ** ATTACK #17 – expected 0x121, REJECTED **
   18  0x123    1  [5E                ]  ** ATTACK #18 – expected 0x121, REJECTED **
   19  0x123    1  [5E                ]  ** ATTACK #19 – expected 0x121, REJECTED **
   20  0x123    1  [5E                ]  ** ATTACK #20 – expected 0x121, REJECTED **
```

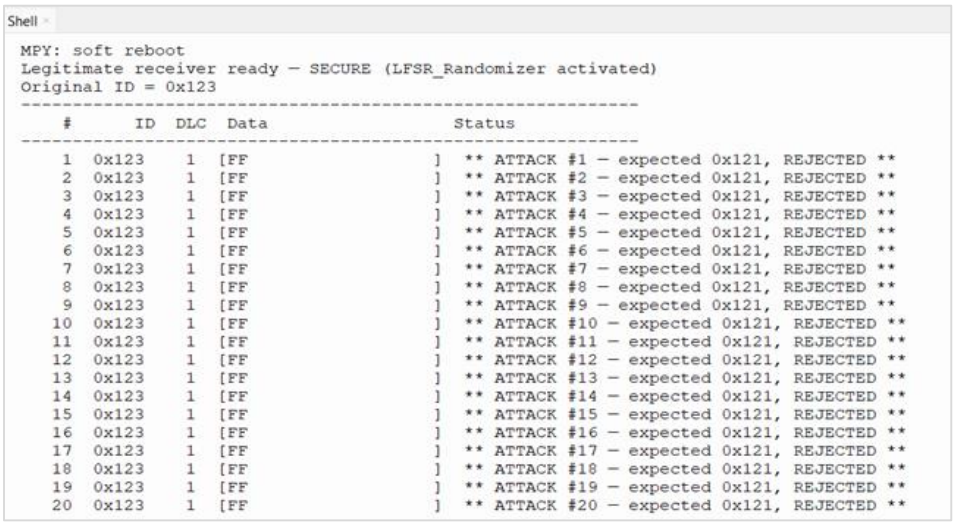
Figure 13. Receiver node in protected mode (LFSR randomizer activated)



```
Shell

MPY: soft reboot
Legitimate receiver ready — PLAIN (LFSR_Randomizer not activated)
Original ID = 0x123
------------------------------------------------------------
    #       ID  DLC  Data                    Status
------------------------------------------------------------
    1   0x123    1  [3A                   ]  OK
    2   0x123    1  [56                   ]  OK
    3   0x123    1  [5E                   ]  OK
    4   0x123    1  [FF                   ]  OK
    5   0x123    1  [FF                   ]  OK
    6   0x123    1  [FF                   ]  OK
    7   0x123    1  [FF                   ]  OK
    8   0x123    1  [FF                   ]  OK
    9   0x123    1  [FF                   ]  OK
   10   0x123    1  [FF                   ]  OK
   11   0x123    1  [FF                   ]  OK
   12   0x123    1  [FF                   ]  OK
   13   0x123    1  [FF                   ]  OK
   14   0x123    1  [FF                   ]  OK
   15   0x123    1  [FF                   ]  OK
   16   0x123    1  [FF                   ]  OK
   17   0x123    1  [FF                   ]  OK
```

Figure 14. Spoofing attack scenario in non-protected mode (data = 0xFF)



```
Shell

MPY: soft reboot
Legitimate receiver ready — SECURE (LFSR_Randomizer activated)
Original ID = 0x123
------------------------------------------------------------
    #       ID  DLC  Data                    Status
------------------------------------------------------------
    1   0x123    1  [FF                   ]  ** ATTACK #1 — expected 0x121, REJECTED **
    2   0x123    1  [FF                   ]  ** ATTACK #2 — expected 0x121, REJECTED **
    3   0x123    1  [FF                   ]  ** ATTACK #3 — expected 0x121, REJECTED **
    4   0x123    1  [FF                   ]  ** ATTACK #4 — expected 0x121, REJECTED **
    5   0x123    1  [FF                   ]  ** ATTACK #5 — expected 0x121, REJECTED **
    6   0x123    1  [FF                   ]  ** ATTACK #6 — expected 0x121, REJECTED **
    7   0x123    1  [FF                   ]  ** ATTACK #7 — expected 0x121, REJECTED **
    8   0x123    1  [FF                   ]  ** ATTACK #8 — expected 0x121, REJECTED **
    9   0x123    1  [FF                   ]  ** ATTACK #9 — expected 0x121, REJECTED **
   10   0x123    1  [FF                   ]  ** ATTACK #10 — expected 0x121, REJECTED **
   11   0x123    1  [FF                   ]  ** ATTACK #11 — expected 0x121, REJECTED **
   12   0x123    1  [FF                   ]  ** ATTACK #12 — expected 0x121, REJECTED **
   13   0x123    1  [FF                   ]  ** ATTACK #13 — expected 0x121, REJECTED **
   14   0x123    1  [FF                   ]  ** ATTACK #14 — expected 0x121, REJECTED **
   15   0x123    1  [FF                   ]  ** ATTACK #15 — expected 0x121, REJECTED **
   16   0x123    1  [FF                   ]  ** ATTACK #16 — expected 0x121, REJECTED **
   17   0x123    1  [FF                   ]  ** ATTACK #17 — expected 0x121, REJECTED **
   18   0x123    1  [FF                   ]  ** ATTACK #18 — expected 0x121, REJECTED **
   19   0x123    1  [FF                   ]  ** ATTACK #19 — expected 0x121, REJECTED **
   20   0x123    1  [FF                   ]  ** ATTACK #20 — expected 0x121, REJECTED **
```

Figure 15. Spoofing attack scenario in protected mode

sender authentication and protection against ECU DoS attacks. Their method relies on shared cryptographic keys and session-based anonymous ID generation, ensuring that only authorized ECUs can recognize the transmitted frames. However, the approach requires pre-shared secret keys, synchronization between ECUs and cryptographic hash functions such as HMAC-SHA1. These operations, while secure, impose additional computational load and require time synchronization mechanisms, which may limit their feasibility for resource constrained ECUs. In contrast, our proposed hardware-based randomization mechanism focuses on lightweight implementation of LFSR-generated identifiers without relying on key management schemes. This design ensures real-time determinism andminimal latency overhead, achieving protection against spoofing and replay attacks while maintaining full compatibility with CAN arbitration. Unlike IA-CAN, our approach does not require periodic session refresh or shared key storage, making it more suitable for low-cost ECUs and safety-critical automotive subsystems where determinism and reliability are essential.

As seen in the introduction, Karray *et al*. [15] proposed an efficient protection mechanism for the CAN bus based on identifier randomization, introducing several strategies to mitigate injection, replay and reverse-engineering attacks.

Their work explored both software and hardware-level implementations and evaluated the effectiveness of randomization through entropy and conditional entropy. Taking the existing IA-CAN scheme [28] as a baseline, they proposed four randomization strategies (Equal Intervals, Frequency Intervals, Dynamic Intervals and Arithmetic Masking), each designed to enhance the unpredictability of CAN identifiers. Our proposed solution shares several conceptual similarities with their approach, where both aim to preserve the arbitration properties of CAN while introducing randomness into message identifiers to prevent spoofing and replay attacks, and both rely on entropy-based statistical analysis to quantify unpredictability and validate the randomization process. However, whereas their design modifies the CAN controller at the protocol level and depends on predefined mathematical mappings, the current solution adopts a lightweight FPGA-based design centred on a PRNG LFSR, allowing the randomization to be implemented externally without impacting existing CAN transceivers or ECUs and ensuring flexible, easy integration into legacy systems.

Furthermore, our approach evaluates randomness using Shannon entropy and a Chi-square goodness-of-fit test for uniformity, whereas Karray *et al*. [15] rely on entropy and conditional entropy. These evaluations are complementary rather than directly comparable because conditional entropy quantifies an attacker's predictive advantage given the original identifier, while our uniformity testing characterizes the statistical distribution of the randomized identifiers. Consequently, while their framework offered valuable theoretical insights and foundational models for entropy-based CAN protection, our proposed design advances this concept by providing a practical and lightweight hardware randomization solution that ensures compatibility, real-time performance and low resource utilization which are key factors for deployment in embedded automotive systems.

SCAN-C [29] is a 64-bit block cipher with a 160-bit key and twelve iterative rounds, combining a Feistel structure with a substitution–permutation network to provide message confidentiality and integrity. SCAN-C protects the message payload through encryption, whereas our mechanism operates at the identifier level to obfuscate CAN IDs and defeat spoofing and replay. The two are therefore complementary rather than substitutes, SCAN-C additionally protects against eavesdropping and payload tampering, which identifier randomization does not address. Securing each frame with SCAN-C requires a full twelve-round transformation which is equivalent to 2,459 cycles and ≈166 µs per operation on an 8-bit microcontroller, plus the complexity related to secure key-management infrastructure. Our randomizer reduces the per-frame operation to a single XOR of the identifier with the LFSR keystream, completing in one clock cycle (~10 ns at 100 MHz) with only 15 LUTs and 23 flip-flops and no cryptographic key storage or distribution. Because frame acceptance is a lightweight comparison rather than a decryption, our scheme also imposes far less computation on the receiver and is better suited to hard real-time and low-cost ECUs. In short, SCAN-C offers stronger, confidentiality-level protection at a higher computational and key-management cost, whereas our mechanism provides targeted, key-free protection against identifier-based attacks at near-zero latency and bus overhead.

To consolidate the preceding discussion, **Table 2** provides a structured comparison between the proposed mechanism and the representative approaches described above.

4. Conclusion and Future Work

In this work, we addressed one of the major security weaknesses of the CAN protocol, namely its vulnerability to spoofing and replay attacks due to the static and predictable nature of CAN identifiers. To mitigate this issue, we proposed a lightweight Verilog-based CAN ID randomization mechanism that introduces unpredictability in the identifier field while preserving the arbitration rules that are essential to the real-time operation of automotive communication. The design was implemented in synthesizable Verilog and successfully tested in Vivado, with simulation waveforms confirming its functional correctness.

To validate the quality of the generated identifiers, we performed a set of statistical randomness analyses. The results highlight that the mechanism provides a strong degree of randomness while maintaining compliance with CAN arbitration requirements.

Table 2. Comparison of the proposed identifier-randomization mechanism with representative CAN security approaches

Criterion	This work	IA-CAN [28]	Karray *et al.* [15]	SCAN-C [29]
Computational overhead	Very low (1 XOR per clock cycle)	High (Keyed HMAC-SHA1 + XOR per frame)	Low (PRNG + mapping function)	High (12-round block cipher per message)
Latency	~10 ns (1 clock @ 100 MHz)	≈12 ms (8-bit MCU); <25 µs (ARM Cortex-M3)	Not quantified in hardware	~166 µs (key gen + enc/dec, ATmega328 @16 MHz)
Implementation complexity	Low (external add-on, no ECU change)	High (keys + time synchronization)	Moderate (controller-level integration)	High (full cipher + key management)

From a hardware perspective, synthesis on a Xilinx Artix-7 FPGA (XC7A35T-1CPG236C) showed that the design requires only 15 Slice LUTs and 23 flip-flops (less than 0.1% of available resources) and reaches a maximum operating frequency of 482 MHz, making it suitable for integration in resource-constrained environments such as automotive ECUs.

Beyond simulation and synthesis, the mechanism was further evaluated on a physical testbed showing that the receiver rejected all injected replay and spoofing frames, without introducing additional bus traffic and with negligible timing overhead.

While the results are promising, the current study has several limitations. First, the randomization source is based on an LFSR, which is not cryptographically secure on its own and could, in principle, be reconstructed by an adversary observing enough consecutive identifiers, as discussed in the LFSR predictability analysis. Second, although the mechanism was validated on a physical testbed, this prototype does not reproduce the full complexity of an in-vehicle network.

Future work will therefore focus on several directions. First, we plan to extend the current physical testbed to a full hardware-in-the-loop setup with realistic in-vehicle traffic to evaluate its impact on latency, error handling and arbitration under bus contention, and to assess the operational reseeding strategies. Finally, we will explore implementing the solution as a dedicated ASIC or FPGA IP core, enabling direct integration into automotive microcontrollers and ensuring scalability for future high-speed in-vehicle CAN variants such as CAN-FD.

In summary, this work provides a step toward enhancing CAN security at the identifier level. By combining a lightweight hardware design with statistical validation and a hardware proof-of-concept, our results indicate that identifier randomization can raise the difficulty of ID-level spoofing and replay attacks without compromising real-time performance, while acknowledging that it does not by itself provide cryptographic-grade confidentiality.

Author's Declaration

Authors' contributions and responsibilities

The authors made substantial contributions to the conception and design of the study. The authors took responsibility for data analysis, interpretation and discussion of results. The authors read and approved the final manuscript.

Funding

This research was undertaken without the support of any external funding.

Availability of data and materials

All data are available from the authors.

Competing interests

The authors declare no competing interest.

Additional information

No additional information from the authors.

Generative AI statement

The author(s) declare that no generative AI or AI-assisted technologies were used in the preparation of this manuscript.

References

[1] Bosch GmbH, "CAN Specification Version 2.0," 1991.

[2] International Organization for Standardization, "ISO 11898-1:2003 — Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," 2003.

[3] F. Hartwich, "CAN with flexible data-rate," Hambach Castle: 13th International CAN Conference (iCC), 2012.

[4] K. Koscher *et al.*, "Experimental Security Analysis of a Modern Automobile," in *2010 IEEE Symposium on Security and Privacy*, IEEE, 2010, pp. 447–462. doi: 10.1109/SP.2010.34.

[5] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," Usenix. [Online]. Available: https://www.usenix.org/conference/usenix-security-11/comprehensive-experimental-analyses-automotive-attack-surfaces

[6] J. Petit and S. E. Shladover, "Potential Cyberattacks on Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2014, doi: 10.1109/TITS.2014.2342271.

[7] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," Las Vegas: Black Hat USA, 2015.

[8] 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*, pp. 1–14, 2014, doi: 10.1109/TITS.2014.2351612.

[9] K.-T. Cho and K. G. Shin, "Error Handling of In-vehicle Networks Makes Them Vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: ACM, Oct. 2016, pp. 1044–1055. doi: 10.1145/2976749.2978302.

[10] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, Jan. 2011, doi: 10.1016/j.ress.2010.06.026.

[11] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla from wireless to CAN bus," Black Hat USA, 2017, pp. 1–16.

[12] W. Xu, S. Yang, and X. Yan, "A Lightweight Intrusion Detection System for In-Vehicle Bus Networks," in *2024 4th International Conference on Intelligent Communications and Computing (ICICC)*, IEEE, Oct. 2024, pp. 149–153. doi: 10.1109/ICICC63565.2024.10780754.

[13] M.-J. Kang and J.-W. Kang, "Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security," *PLOS ONE*, vol. 11, no. 6, p. e0155781, Jun. 2016, doi: 10.1371/journal.pone.0155781.

[14] S. Khandelwal and S. Shreejith, "A Lightweight FPGA-based IDS-ECU Architecture for Automotive CAN," in *2022 International Conference on Field-Programmable Technology (ICFPT)*, IEEE, Dec. 2022, pp. 1–9. doi: 10.1109/ICFPT56656.2022.9974508.

[15] K. Karray, J.-L. Danger, S. Guilley, and M. A. Elaabid, "Identifier Randomization: An Efficient Protection Against CAN-Bus Attacks," in *Cyber-Physical Systems Security*, Cham: Springer International Publishing, 2018, pp. 219–254. doi: 10.1007/978-3-319-98935-8_11.

[16] B. Lampe and W. Meng, "Intrusion Detection in the Automotive Domain: A Comprehensive Review," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2356–2426, 2023, doi: 10.1109/COMST.2023.3309864.

[17] AUTOSAR Consortium, "Specification of Secure Onboard Communication (SecOC)," 2017.

[18] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and Countermeasures for In-Vehicle Networks," *ACM Computing Surveys*, vol. 54, no. 1, pp. 1–37, Jan. 2022, doi: 10.1145/3431233.

[19] G. I. Mary, Z. C. Alex, and L. Jenkins, "Response Time Analysis of Messages in Controller Area Network: A Review," *Journal of Computer Networks and Communications*, vol. 2013, pp. 1–11, 2013, doi: 10.1155/2013/148015.

[20] E. Avaroğlu, İ. Koyuncu, A. B. Özer, and M. Türk, "Hybrid pseudo-random number generator for cryptographic systems," *Nonlinear Dynamics*, vol. 82, no. 1–2, pp. 239–248, Oct. 2015, doi: 10.1007/s11071-015-2152-8.

[21] P. Chandravanshi, J. K. Meka, V. Mongia, R. P. Singh, and S. Prabhakar, "LFSR based RNG on low-cost FPGA for QKD applications," 2023, *arXiv*. doi: arXiv:2307.16431.

[22] C. E. Shannon, *The Mathematical Theory of Communication*. Urbana, IL, USA: University of Illinois Press, 1949.

[23] Y. Wu, J. P. Noonan, and S. Agaian, "Shannon entropy based randomness measurement and test for image encryption," 2011. doi: 10.48550/arXiv.1103.5520.

[24] L. E. Bassham *et al.*, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Gaithersburg, MD, MD, 2010. doi: 10.6028/NIST.SP.800-22r1a.

[25] P. Kietzmann, T. C. Schmidt, and M. Wählisch, "A Guideline on Pseudorandom Number Generation (PRNG) in the IoT," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–38, Jul. 2022, doi: 10.1145/3453159.

[26] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, Jan. 1969, doi: 10.1109/TIT.1969.1054260.

[27] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 1996.

[28] K. Han, A. Weimerskirch, and K. G. Shin, "A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier," in *European Embedded Security in Cars (ESCAR)*, 2015, pp. 13–29.

[29] N. Hediyal, B. P. Divakar, and K. Narayanaswamy, "SCAN-C: a lightweight cryptographic algorithm to secure CAN communications in modern vehicles," *Cybersecurity*, vol. 8, no. 1, p. 49, Jul. 2025, doi: 10.1186/s42400-024-00291-z.