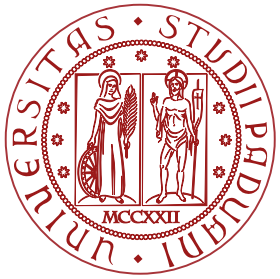


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DEPARTMENT OF MATHEMATICS "TULLIO LEVI-CIVITA"

Master Degree in Mathematics

LCAP - A Lightweight CAN Authentication Protocol for Practical Automotive Development

Master Candidate:
Prasanth Mohan

Student ID: 2078711

Supervisor:
Prof. Mauro Conti

Co-supervisor:
Dr. Harsha Vasudev

Academic Year 2025/2026

Contents

List of Figures

List of Tables

Abstract

Over the past two decades, passenger vehicles have transitioned from isolated mechanical systems into highly connected cyber-physical networks. While innovations such as telematics, vehicle-to-everything (V2X) communication, and autonomous driving have vastly improved functionality, they have simultaneously expanded the automotive digital attack surface, introducing severe security vulnerabilities. At the core of this issue is the Classical Controller Area Network (CAN 2.0B) the standard backbone for in-vehicle communication which entirely lacks fundamental security controls like source authentication, integrity verification, and replay protection. Securing this legacy architecture remains a notorious challenge due to its strict 8-byte payload limitation. Traditional multi frame authentication schemes regularly fail to meet stringent automotive requirements; by splitting security data across multiple messages, they clog the bus and introduce unpredictable transmission delays, known as scheduling jitter.

To resolve these constraints, this thesis introduces the Lightweight CAN Authentication Protocol (LCAP), a deterministic, single-frame authentication mechanism tailored for resource-constrained legacy Electronic Control Units (ECUs). LCAP fits seamlessly within the standard 64-bit CAN data field by strategically partitioning the payload into 24 bits for application data, 24 bits for a monotonic freshness counter (Ctr_{24}), and 16 bits for a truncated authentication tag (Tag_{16}) computed via HMAC-SHA256. To guarantee network availability and reliability in electrically noisy automotive environments, the receiver utilizes a stateful Finite State Machine (FSM) backed by a sliding soft-resynchronization window ($W = 16$). This design allows the system to gracefully tolerate brief frame drops due to transient noise without triggering heavy, bus-loading resynchronization routines.

The protocol was empirically validated on a physical hardware-in-the-loop (HIL) testbed leveraging 8-bit ATmega328P microcontrollers and standalone MCP2515 CAN controllers. Benchmarks demonstrate a frame preparation time of 405 μs on the transmitter side and a verification time of 418 μs on the receiver side. This keeps the entire processing loop well under 1.5 ms while demanding just 11.42% CPU utilization at a safety-critical transmission frequency of 100 Hz, confirming LCAP's viability for low-power embedded hardware. Furthermore, our security analysis indicates that while truncating the HMAC tag naturally reduces static entropy, online brute-force attacks remain practically infeasible. Because the physical bandwidth of the CAN bus limits frame injection speeds, and our Adaptive Rate Limiting (ARL) policy aggressively flags rapid errors, an attacker is forced into a noisy, highly detectable window—stretching the expected time to compromise to 16.78 seconds at 250 kbps. Ultimately, LCAP proves that single-frame determinism can successfully safeguard legacy vehicle networks without compromising real-time safety or strict timing constraints.

Chapter 1

Introduction

1.1 Vehicular Digital Revolution

For much of the twentieth century, the automotive industry was mainly shaped by mechanical engineering. The quality of a vehicle was often judged by the reliability of its engine, the strength of its chassis, the smoothness of its transmission, and the precision of its mechanical control systems. Over the last two decades, however, this view of the automobile has changed considerably. Modern vehicles are no longer only mechanical machines. They have become complex **Cyber Physical Systems (CPS)**, where physical motion, electronic control, embedded computation, and communication networks operate together. In this sense, the vehicle has become one of the most visible examples of computing moving from traditional office or industrial environments to everyday physical systems.

In older vehicles, especially those of the 1990s, electronic control was relatively limited. A small number of controllers were used for functions such as fuel injection, ignition timing, or basic diagnostics. These systems were important, but they were still mostly isolated and narrow in purpose. By contrast, a modern high-end vehicle may contain between 70 and 100 Electronic Control Units (ECUs). These ECUs form a distributed embedded network inside the vehicle and collectively manage a wide range of functions. Some of these functions are critical for safety, such as powertrain control, Anti lock brake systems (ABS), Electronic Stability Control (ESC), steering support, and braking assistance. Other are related to comfort or convenience, such as information technology, climate control, lighting, seat adjustment, and driver personalization.

This transformation has been driven by several factors. Customers expect vehicles to provide more safety, comfort, automation, and connectivity than before. At the same time, manufacturers are under pressure to support Advanced Driver Assistance Systems (ADAS), autonomous driving functions, and Vehicle-to-Everything (V2X) communication. As a result, the modern vehicle is no longer an isolated mechanical object. It is increasingly connected to external networks, mobile devices, cloud services, road infrastructure, and other vehicles. This connectivity brings clear benefits, including better navigation, remote diagnostics, software updates, and improved driver assistance. However, it also introduces a much larger attack surface. The same interfaces that make a vehicle smarter and more connected can also provide entry points for attackers. The

boundary between the digital and physical world has therefore become a major concern in automotive security because a digital compromise can ultimately influence steering, braking, acceleration, and other physical actions.

1.2 History of In-Vehicle Insecurity

For many years, the security of vehicle networks has received limited attention compared to their reliability and real-time performance. Automotive communication systems were designed primarily to be robust, predictable, low-cost, and suitable for harsh electrical environments. Security was not treated as a central design requirement because it was traditionally assumed that vehicles were closed systems. In other words, if an attacker did not have physical access to the vehicle wiring or diagnostic port, the internal network was considered difficult to reach.

This assumption began to break down as the vehicles connected. A turning point came in 2015 with the well-known Jeep Cherokee attack demonstrated by Charlie Miller and Chris Valasek. In that work, the researchers exploited a vulnerability in the Harman Uconnect infotainment system. From this external entry point, they were able to move from the cellular-connected head unit into the internal Controller Area Network (CAN) bus. Once they reached the CAN bus, they could inject messages that affected vehicle behavior. The demonstration included actions such as controlling parts of the information system, interfering with the engine, and influencing braking or steering behavior under certain conditions.

The importance of this attack was not only in the technical details of the exploit, but also in what it revealed about the design philosophy of older automotive networks. The internal network of the vehicle had been built around trust. ECUs on the CAN bus were generally expected to behave honestly, and messages were accepted mainly because they appeared on the bus with the correct identifier. The original CAN design came from a period when remote connectivity was not part of the vehicle architecture. Engineers reasonably focused on reliability, arbitration, timing, and noise immunity, rather than cryptographic authentication.

In today's environment, that assumption is no longer sufficient. A vehicle may include cellular modems, Bluetooth interfaces, Wi-Fi connectivity, USB ports, telematics units, mobile application interfaces, and even V2X communication modules. If an attacker compromises one externally reachable component, that component may become a bridge into the internal vehicle network. Once inside, the attacker may attempt to send malicious CAN frames that imitate legitimate control messages. This is why the problem is serious: a weakness in a seemingly non-critical subsystem, such as infotainment or telematics, may create risk for safety-critical ECUs such as braking, steering, or engine control.

1.3 Security Gap in Classical CAN

The Controller Area Network, standardized as ISO 11898, was originally developed by Robert Bosch GmbH in the 1980s. It was created to reduce wiring complexity, improve communication reliability, and provide deterministic message arbitration in automotive environments. From an engineering perspective, CAN has been extremely successful.

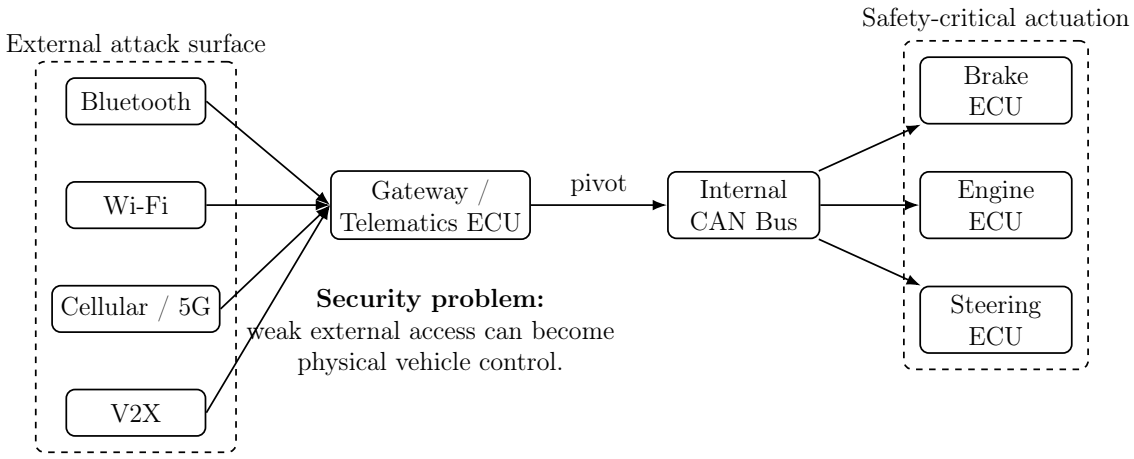


Figure 1.1: Attack surface of a connected vehicle: external interfaces provide entry points that may allow an attacker to pivot toward the internal CAN bus and safety-critical ECUs.

It is inexpensive, robust to electrical noise and is well suited for real-time embedded communication. These strengths explain why CAN is still widely used in vehicles today.

However, CAN was not designed with modern cybersecurity threats in mind. Its basic communication model is broadcast-based, meaning that every node on the bus can observe transmitted frames. In addition, the CAN identifier mainly describes the priority and meaning of the message, not the identity of the sender. As a result, Classical CAN lacks several properties that are normally expected in secure communication systems.

1. **Source Authentication:** A standard CAN frame does not contain a cryptographic field that proves which ECU generated the message. If a compromised or malicious node transmits a frame with the same identifier as a legitimate ECU, other nodes may not be able to distinguish the forged frame from the real one. In CAN, the message ID is usually related to message content and priority, not to a verified sender identity. This creates a serious weakness in a system where different ECUs share the same communication medium.

2. **Data Integrity:** CAN includes a Cyclic Redundancy Check (CRC), but the purpose of this CRC is to detect accidental transmission errors, not malicious modification. It is effective against random bit errors caused by electrical noise, electromagnetic interference, or physical-layer disturbances. However, it is not a cryptographic integrity mechanism. Since the CRC algorithm is public and does not use a secret key, an attacker who constructs a malicious payload can also compute a valid CRC for that payload.

3. **Freshness and Replay Resistance:** Classical CAN frames do not include timestamps, nonces, counters, or sequence numbers. This means that a valid frame captured at one time may be replayed later. For example, an attacker could record a legitimate command and transmit it again when the vehicle is in a different state. Without a freshness mechanism, the receiver has no reliable way to know whether a frame is new or simply an old message being repeated.

These limitations do not mean that CAN was poorly designed. Rather, they show that CAN was designed for a different threat model. In the original environment, the main concern was reliable real-time communication between trusted ECUs. In modern connected vehicles, that assumption is no longer enough. The challenge is therefore to add meaningful security protection without destroying the timing, bandwidth, and compatibility properties that made CAN useful in the first place.

1.4 The 8-Byte Constraint

Securing Classical CAN is difficult because the available payload space is very small. A standard Classical CAN data frame can carry at most 8 bytes, or 64 bits, of application data. This is a strict limitation. Any security mechanism must either fit inside this payload or use additional frames. Both choices create trade-offs.

From a mathematical point of view, the problem can be understood as a constrained optimization problem. The protocol must allocate only 64 bits among application data, freshness information, and authentication information. If too many bits are reserved for security metadata, the useful application data becomes too small. If too many bits are reserved for application data, the security strength becomes weak. This balance is the central design problem addressed in this thesis.

Many conventional cryptographic mechanisms are too large for Classical CAN. For example:

- **Digital Signatures:** An ECDSA signature typically requires 64 bytes. Transmitting such a signature over Classical CAN would require multiple frames, even before considering the actual sensor or control data. This would significantly increase bus utilization and delay.

- **Standard MACs:** A full HMAC-SHA256 output is 32 bytes or 256 bits. Sending the complete tag would require four full CAN payloads. This is not practical for short periodic messages that must be transmitted within strict real-time deadlines.

One possible solution is to fragment the authentication data into several CAN frames. However, this approach introduces new problems. Multi-frame authentication increases bus load, requires buffering at the receiver, and creates additional timing uncertainty. In real-time automotive systems, this uncertainty is not a minor inconvenience. A delay of only a few milliseconds may matter for control loops related to braking, steering, or engine management.

This is where the problem becomes especially important for embedded and automotive systems. Security cannot be added as if the CAN bus were a high-bandwidth computer network. A vehicle control network has strict timing and scheduling requirements. If authentication creates too much delay, then the security mechanism itself may reduce system safety. Therefore, the goal is not simply to maximize the cryptographic strength in isolation. The goal is to design a security mechanism that provides useful authenticity and freshness while preserving timing predictability and keeping the frame structure compatible with Classical CAN.

1.5 Research Aim and Objectives

The main aim of this thesis is to design, implement, and evaluate *LCAP* (Lightweight CAN Authentication Protocol), a lightweight authentication mechanism for classic CAN networks. The motivation behind this work is the need for a security solution that can be applied to legacy and constrained automotive environments. Many vehicles currently in use still depend on Classical CAN and resource-limited ECUs. Replacing the entire global vehicle fleet with newer architectures is not realistic in the short term. Therefore, practical protection mechanisms must work within the limitations of existing systems.

The proposed protocol is based on a simple idea: authentication and freshness information should be placed inside a single CAN data frame whenever possible. This avoids extra authentication frames and preserves the timing behavior of the original message stream. At the same time, the protocol must provide enough protection to make spoofing and replay attacks significantly harder in practice.

To achieve this, the project focuses on several key objectives:

- **Payload Partitioning Optimization:** The first objective is to define a practical division of the 64-bit Classical CAN payload. The payload must carry application data, a freshness counter, and an authentication tag without requiring frame fragmentation.

- **Resiliency and State Management:** The second objective is to design receiver-side logic that can handle ordinary frame loss without immediately rejecting all future messages. In a real vehicle, frames may be lost because of electromagnetic interference or temporary bus disturbance. The protocol must therefore distinguish between natural communication loss and suspicious replay or forgery attempts.

- **Probabilistic Security Analysis:** The third objective is to analyze the security of a truncated authentication tag in the physical context of the CAN bus. Since a 16-bit tag is much smaller than a standard cryptographic tag, its practical security must be studied together with CAN timing, bus capacity, and rate-limiting assumptions.

- **Empirical Validation:** The fourth objective is to implement the protocol on low-cost 8-bit microcontroller hardware and evaluate its performance. This includes measuring latency, observing receiver behavior, and checking whether the authentication process is feasible on resource-constrained platforms.

1.6 Thesis Contributions

This thesis contributes a lightweight and practical approach to CAN message authentication. The first contribution is the formal definition of the *LCAP* frame structure, where application data, a 24-bit freshness counter, and a 16-bit truncated authentication tag are packed into a single 64-bit Classical CAN payload. This structure is designed to preserve compatibility with existing CAN frames while adding authentication and replay resistance.

The second contribution is the receiver-side state logic, including the use of a soft-resynchronization mechanism. This mechanism allows the receiver to remain robust when a small number of frames are lost, while still rejecting stale messages and frames with invalid authentication tags. This is important because availability is a safety requirement in automotive systems, not only a convenience.

The third contribution is a probabilistic security analysis of the truncated tag in the specific context of CAN communication. Instead of evaluating the 16-bit tag only in a general computing environment, the analysis considers the physical transmission rate of CAN and the limited number of online attempts an attacker can make without creating noticeable bus disturbance.

The fourth contribution is the implementation of the proposed protocol on an Arduino-based hardware-in-the-loop prototype using MCP2515 CAN controller modules. This prototype demonstrates that the protocol can run on constrained hardware and provides a practical basis for evaluating latency, payload construction, and receiver verification behavior.

Overall, this work attempts to bridge the gap between theoretical cryptography and practical automotive engineering. It does not claim that a small truncated tag provides the same security level as full-length authentication codes used in high-bandwidth networks. Instead, it argues that carefully designed lightweight authentication can still provide meaningful protection for Classical CAN systems, especially when combined with freshness checking, rate awareness, and deterministic single-frame operation.

Chapter 2

Background and Preliminaries

Before designing a security protocol for vehicles, it is necessary to understand the environment in which the protocol will operate. Automotive networks are not the same as ordinary computer networks. They are strongly constrained by timing, bandwidth, electrical noise, hardware cost, and long product lifetimes. A solution that looks reasonable in a desktop or internet setting may be completely unsuitable inside a vehicle. For this reason, this chapter introduces the technical background needed for the design of *LCAP*.

The chapter first explains the Controller Area Network (CAN), which is the communication bus targeted in this thesis. It then discusses the physical behavior of CAN, including dominant and recessive signaling, arbitration, synchronization, and bit-stuffing. These details are important because the security protocol must respect the real-time behavior of the bus. Finally, the chapter introduces the cryptographic concepts used in the proposed design, especially symmetric-key authentication and truncated HMAC tags.

2.1 The Controller Area Network (CAN) Protocol

The Controller Area Network (CAN), originally standardized under ISO 11898-1, is a multi-master, broadcast, serial communication bus. It was designed for embedded environments where several electronic modules need to exchange short messages reliably and with predictable timing. In the automotive domain, CAN became popular because it reduced wiring complexity while still providing robust communication in a noisy electrical environment.

Unlike traditional computer networks such as Ethernet, CAN does not use destination addresses in the same way. In Ethernet, a frame is normally sent from one device to another device using Media Access Control (MAC) addresses. CAN works differently. It is fundamentally **content-addressed**. This means that a CAN frame is identified mainly by the meaning and priority of the message, not by the address of the receiver.

In this architecture, every transmitted message is broadcast to all nodes connected to the bus. Each message begins with an Identifier (ID), and this identifier describes the type of information carried by the frame. For example, in a vehicle network, ID `0x123` may be assigned to an engine-speed signal, while ID `0x456` may correspond to wheel-speed information. A receiving ECU decides whether to process or ignore a frame based on the identifier and its own local configuration.

This design makes CAN simple and flexible. New nodes can listen to existing messages without requiring the sender to know exactly which receivers exist. At the same time, it creates an important security weakness. Since the identifier describes the message type rather than the authenticated sender, a malicious or compromised node can attempt to transmit a frame using an identifier normally associated with another ECU. This is one of the reasons why authentication is difficult in Classical CAN.

2.2 Physical Layer Mechanics

The CAN bus uses a differential signaling mechanism with two wires: CAN-High (CANH) and CAN-Low (CANL). Instead of representing bits using the voltage of a single wire, CAN represents bus states through the voltage difference between these two wires. This design is especially useful in vehicles, where motors, ignition systems, alternators, and switching circuits can generate electromagnetic interference.

Differential signaling improves noise immunity because external noise often affects both wires in a similar way. Since the receiver is mainly interested in the difference between CANH and CANL, common-mode noise can be rejected more effectively. This is one of the reasons CAN has remained reliable in harsh automotive environments.

2.2.1 Dominant and Recessive States

The logical behavior of the CAN bus is based on two states: recessive and dominant. These states are defined by the voltage difference between CANH and CANL.

- **Recessive (Logical 1):** In the recessive state, the transceiver does not actively force a strong differential voltage onto the bus. Both CANH and CANL remain close to a common bias voltage, typically around 2.5V. Since the two wires are at nearly the same voltage, the differential voltage $V_{diff} = V_{CANH} - V_{CANL}$ is approximately 0V.

- **Dominant (Logical 0):** In the dominant state, the transceiver actively drives the bus. CANH is driven upward, typically toward about 3.5V, while CANL is driven downward, typically toward about 1.5V. This creates a differential voltage of approximately 2.0V. The dominant state therefore represents a stronger electrical condition than the recessive state.

The terms “dominant” and “recessive” are important because they describe not only voltage levels, but also the arbitration behavior of the bus. If one node transmits a dominant bit while another node transmits a recessive bit, the observed bus state becomes dominant. In practice, this means that logical 0 overrides logical 1.

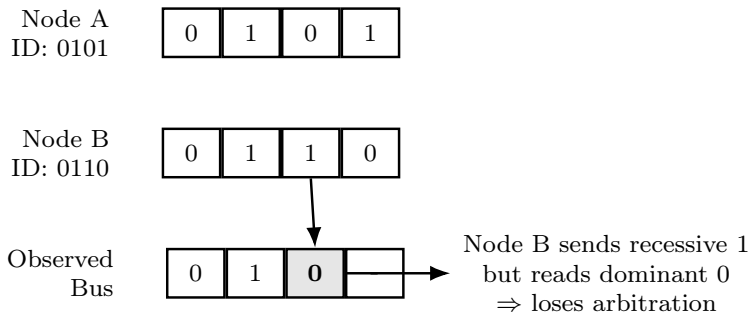


Figure 2.1: Non-destructive CAN arbitration. When two ECUs transmit simultaneously, the dominant bit forces the bus state, allowing the lower numerical identifier to retain priority.

Mathematically, this behavior can be understood as a **Wired-AND** mechanism. If multiple nodes are transmitting at the same time, and at least one of them transmits a dominant 0, the resulting bus value is 0. The bus becomes recessive 1 only when all transmitting nodes send recessive bits. This simple physical rule is the basis for CAN arbitration.

2.2.2 Bitwise Arbitration: The Priority Mechanism

The Wired-AND property allows CAN to perform non-destructive arbitration. This means that if two or more ECUs begin transmitting at the same time, the network can resolve the conflict without corrupting the winning frame. The mechanism works bit by bit during the identifier field.

When the bus is idle, any node may begin transmission. If two nodes start at nearly the same time, both begin by transmitting their identifier bits. At each bit position, every transmitting node also monitors the actual state of the bus. If a node transmits a recessive 1 but reads back a dominant 0, it knows that another node is transmitting a higher-priority message. The losing node immediately stops transmitting and waits until the bus becomes idle again.

Because dominant 0 overrides recessive 1, the message with the numerically lower identifier wins arbitration. This gives CAN its priority-based communication behavior. Safety-critical messages can be assigned lower identifiers so that they gain access to the bus before less urgent messages.

This arbitration method is extremely useful for real-time automotive control. It avoids random collisions and retransmission delays of the kind seen in some other communication systems. More importantly, the winning frame is not destroyed during arbitration. The highest-priority message continues transmission normally, which helps preserve deterministic timing for safety-critical signals.

2.3 Clock Synchronization and Bit-Stuffing

CAN is an asynchronous protocol, which means that nodes do not share one global clock signal. Each node has its own local oscillator, and small differences between these oscillators can cause timing drift. To maintain correct sampling of bits, CAN nodes must continuously resynchronize based on signal transitions observed on the bus.

This requirement creates a problem when a long sequence of identical bits is transmitted. If the bus remains at the same logical level for too long, there may not be enough edges for receivers to adjust their timing. To solve this, CAN uses a mechanism known as **Bit-Stuffing**.

2.3.1 The Stuffing Rule

The bit-stuffing rule is simple. If the CAN controller detects five consecutive bits with the same logical value, it automatically inserts one additional bit of the opposite value. For example, after five consecutive dominant bits, a recessive stuff bit is inserted. Similarly, after five consecutive recessive bits, a dominant stuff bit is inserted.

This inserted bit is not part of the original payload. It is added by the transmitter for synchronization purposes and removed by the receiver before the frame is interpreted. From the point of view of the application data, bit-stuffing is invisible. From the point of view of timing, however, it is important because it changes the actual number of bits transmitted on the wire.

This means that two CAN frames with the same nominal structure may not take exactly the same time to transmit. Their transmission time depends partly on the bit patterns inside the frame. A payload containing long runs of identical bits may require more stuff bits than a payload with frequent transitions.

2.3.2 Mathematical Impact on Frame Length

From a timing-analysis perspective, bit-stuffing introduces variability into the frame length. This matters because real-time systems must often be analyzed using worst-case timing assumptions. If B denotes the number of original bits in a frame, then the number of transmitted bits after stuffing is larger than B whenever stuff bits are inserted.

A simplified way to express the relationship is through the following inequality:

$$B + \lfloor \frac{B-1}{4} \rfloor \leq N_{total} \leq B + \lfloor \frac{B-1}{4} \rfloor + \text{Stuff\_Bits}_{max} \tag{2.1}$$

Here, N_{total} represents the total number of bits placed on the wire after the stuffing process. The exact value depends on the data pattern and on where long runs of identical bits occur. For an $LCAP$ frame carrying 8 bytes of data, the maximum theoretical frame size is approximately 135 bits when overhead and stuffing are considered.

At a bus speed of 500 kbps, each bit takes 2 microseconds to transmit. Therefore, even a small variation in the number of transmitted bits can affect the final time-on-wire. This is important for **Worst-Case Execution Time (WCET)** analysis because the security layer must not disturb the predictable timing behavior expected in an automotive control network.

2.4 The Anatomy of a Classical CAN Frame

A Classical CAN 2.0B frame is compact and carefully structured. For the design of *LCAP*, the most important question is where security information can be placed without changing the frame format or requiring additional frames. Since Classical CAN provides only up to 64 bits of data payload, the data field becomes the central constraint.

The main fields of a Classical CAN frame are as follows:

1. **SOF (1 bit):** The Start of Frame bit marks the beginning of a CAN transmission. It signals to other nodes that a new frame is starting.

2. **Arbitration Field (12 bits):** This field contains the 11-bit identifier and the RTR (Remote Transmission Request) bit. The identifier is also used during arbitration, so it determines message priority on the bus.

3. **Control Field (6 bits):** The control field includes information such as the Data Length Code (DLC), which specifies the number of data bytes carried in the frame. In Classical CAN, this value ranges from 0 to 8 bytes.

4. **Data Field (0–64 bits):** This field carries the application payload. For *LCAP*, this is the most important part of the frame because it is the only practical location for application data, freshness information, and authentication metadata.

5. **CRC Field (16 bits):** The Cyclic Redundancy Check field is used for error detection at the communication level. It helps detect accidental transmission errors, but it does not provide cryptographic protection against malicious modification.

6. **ACK Field (2 bits):** The acknowledgement field allows receiving nodes to signal that the frame was received correctly at the CAN level.

7. **EOF (7 bits):** The End of Frame field marks the end of the CAN frame.

The structure shows why security is difficult in Classical CAN. There is no dedicated field for sender identity, cryptographic authentication, freshness counters, or replay protection. Therefore, any backward-compatible security mechanism must use the data field carefully.

2.5 Cryptographic Foundations for Embedded Systems

The cryptographic choices in this thesis are shaped by the limitations of embedded automotive hardware. Many legacy ECUs have small memories, low clock speeds, and limited arithmetic capability. A security protocol that requires large buffers, long keys, expensive modular arithmetic, or multiple message exchanges may not be suitable for this environment.

For this reason, this thesis focuses on **Symmetric-Key Cryptography**. In a symmetric-key system, the sender and receiver share the same secret key. The sender uses the key

to compute an authentication value, and the receiver uses the same key to verify it. This approach is far more practical for small embedded systems than public-key cryptography.

Asymmetric algorithms such as RSA or ECC are powerful and widely used in many security systems. However, they usually require operations such as modular exponentiation or elliptic-curve point multiplication. These operations are computationally expensive, especially on an 8-bit microcontroller such as the ATmega328P. In a real-time automotive setting, a delay of hundreds of milliseconds would be unacceptable for many periodic control messages.

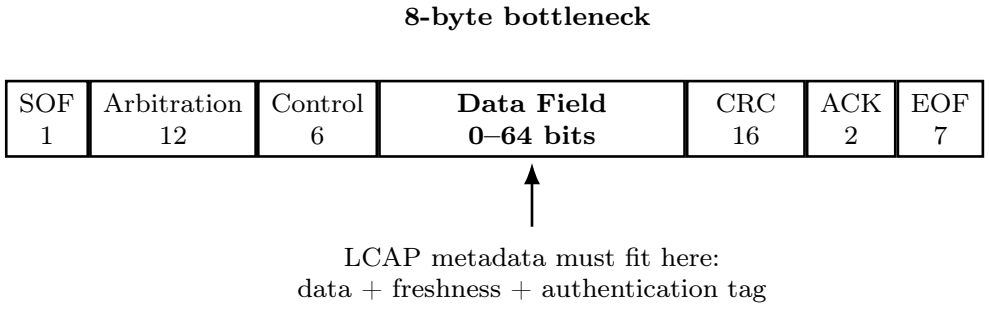


Figure 2.2: Structural anatomy of a Classical CAN 2.0B data frame. The 64-bit data field is the rigid boundary condition that motivates the single-frame design of LCAP.

2.5.1 HMAC-SHA256: The Security Core

The authentication mechanism used in *LCAP* is based on the Hash-based Message Authentication Code (HMAC) construction. HMAC combines a cryptographic hash function with a secret key to produce a message authentication tag. The receiver can recompute the tag using the same key and compare it with the tag received in the frame.

In this thesis, HMAC-SHA256 is used as the main cryptographic primitive. SHA-256 is a well-known hash function, and HMAC-SHA256 is commonly used because it provides strong resistance against several attacks that affect simpler keyed-hash constructions. In particular, the nested structure of HMAC helps prevent length-extension attacks, which are relevant for hash functions built using Merkle-Damgård style constructions.

Internally, SHA-256 performs 64 rounds of operations on 32-bit words. These operations include logical functions such as AND, OR, XOR, bit rotations, shifts, and modular additions. On a 32-bit processor, these operations are natural and efficient. On an 8-bit processor, however, each 32-bit operation must be broken into several 8-bit operations. This increases the number of instructions required and therefore increases the execution time.

For the purposes of this thesis, this limitation is important rather than inconvenient. If the protocol can be made practical on an 8-bit processor, then the design is more likely to be practical on stronger automotive microcontrollers. The implementation must therefore be careful with memory, timing, and data movement so that the cryptographic computation does not dominate the behavior of the ECU.

2.5.2 Truncation and The Probability of Forgery

A full HMAC-SHA256 output is 256 bits long. Classical CAN, however, provides only 64 bits of payload space in total. Since $LCAP$ must also include application data and freshness information, the complete 256-bit authentication value cannot be transmitted. For this reason, the protocol uses a truncation function τ:

$$Tag_{16} = \mathrm{MSB}_{16}(HMAC(K, m)) \tag{2.2}$$

This means that the protocol computes the full HMAC value internally, but only transmits the most significant 16 bits as the authentication tag. The resulting tag has $2^{16} = 65,536$ possible values. In a traditional high-bandwidth computer network, a 16-bit tag would be considered too small. An attacker with a large number of attempts could eventually guess a valid tag.

However, the CAN environment must be evaluated differently. An attacker cannot test guesses at unlimited speed without occupying the physical bus. Every forged attempt must be transmitted as a CAN frame, and each frame consumes time on the bus. If the attacker sends frames too aggressively, the attack itself becomes visible as abnormal bus activity or even a denial-of-service condition. Therefore, the practical security of the truncated tag must be studied together with the physical rate limit of the CAN network.

This thesis returns to this issue in Chapter 8, where the probability of forgery is analyzed in relation to bus speed, frame length, and online attack capacity. The key point at this stage is that $LCAP$ does not treat the 16-bit tag as equivalent to a full-length MAC. Instead, it uses the tag as part of a lightweight security design that also depends on freshness counters, receiver state management, and the timing constraints of the physical bus.

Chapter 3

Literature Survey: The Landscape of Vehicular Security

3.1 Taxonomy of Existing Solutions

The problem of securing the CAN bus has been studied from many different directions. Over the years, researchers have proposed a wide range of methods to add authentication, integrity, and replay protection to in-vehicle networks. Although the details differ from one proposal to another, most of these solutions can be grouped into three broad categories: physical-layer augmentation, hardware-assisted designs, and software-only security mechanisms.

Physical-layer approaches try to use the electrical behavior of the CAN bus itself as an additional communication channel. Instead of placing all security information inside the 8-byte data field, these methods attempt to encode extra information through changes in the physical signal. The main advantage is that the original CAN payload can remain available for application data. However, these methods often require changes to the transceiver or depend on signal properties that may not remain stable in a real vehicle.

Hardware-assisted approaches rely on additional components, secure modules, or modified ECUs to support authentication. These designs can provide strong security because they are not limited only by the original CAN frame structure. In practice, however, they may be difficult to deploy in existing vehicles. Replacing or upgrading vehicle hardware is expensive, and many legacy ECUs were not designed with secure coprocessors or hardware security modules in mind.

Software-only approaches try to improve CAN security without changing the underlying hardware. These are attractive because they can, at least in principle, be deployed through firmware updates or lightweight software modifications. The difficulty is that Classical CAN provides only 8 bytes of payload, so software-only designs must make difficult compromises between application data, freshness information, and authentication metadata. This is the category most closely related to the design of $LCAP$.

3.2 Detailed Critique of Physical-Layer Approaches

3.2.1 CAN-MM (Frequency Modulation)

CAN-MM was proposed as one possible way to address the well-known "8-byte bottleneck" of Classical CAN. The main idea is to avoid placing the Message Authentication Code directly inside the normal digital payload. Instead, the authentication information is moved into the physical layer by using frequency-modulated sidebands. In other words, the CAN data field can continue to carry the original application data, while extra security information is transmitted through carefully controlled physical signal variations.

At first glance, this is an attractive idea. It appears to solve the payload-space problem without reducing the amount of application data carried by each CAN frame. For systems where every byte of payload is important, this kind of approach can seem very promising. It also shows an important direction in vehicular security research: not every security proposal has to work only at the application layer or data-link layer.

Mathematical and Practical Critique: The main weakness of CAN-MM is that it depends heavily on the quality and stability of the physical signal. In a controlled laboratory environment, frequency-modulated sideband information may be measurable and usable. A real vehicle, however, is a much harsher environment. Spark plugs, alternators, electric motors, switching circuits, and long wiring harnesses all contribute to electrical noise. This creates a Signal-to-Noise Ratio (SNR) problem. The authentication information is no longer protected only by mathematical design; it also depends on whether the receiver can reliably distinguish subtle physical-layer variations from environmental noise.

There is also a deployment problem. A solution that requires modified CAN transceivers is not easily backward compatible. Existing vehicles already contain a large number of ECUs and transceiver modules, and replacing these components across the global vehicle fleet would be costly and unrealistic. For this thesis, backward compatibility is a central requirement. The goal is to improve security on legacy CAN systems without demanding new physical-layer hardware. For that reason, although CAN-MM is technically interesting, it does not satisfy the practical deployment constraints that motivate $LCAP$.

3.3 Detailed Critique of Software-Only Approaches

3.3.1 CANAUTH (Multi-Frame Authentication)

CANAUTH is an example of a software-based approach that attempts to add authentication to CAN by distributing a larger MAC across multiple CAN frames. Instead of trying to fit the complete authentication value into one 8-byte frame, the protocol divides the authentication information into several fragments and sends them over a sequence of frames.

This design has a clear security motivation. A longer MAC gives a stronger cryptographic security margin than a very short tag. In a normal computer network, sending a larger authentication value is not usually a major problem. In Classical CAN, however, every additional frame consumes bus time and may interfere with real-time communication. For this reason, the use of multiple frames becomes a serious issue in automotive

systems.

Critique - The Stochastic Jitter Problem: From a mathematical and real-time perspective, multi-frame authentication changes the nature of a CAN message. A single sensor update is no longer a single transmission. It becomes a small train of related messages that must be sent, received, and interpreted together. On a lightly loaded bus, this may work acceptably. On a busy vehicle network, however, the fragments can be delayed by higher-priority traffic.

This delay is not always constant. Sometimes the fragments may arrive close together, and sometimes they may be separated by arbitration delays. This creates jitter, meaning variation in message arrival and processing time. In a safety-critical control loop, jitter is not just an inconvenience. It can affect the stability and responsiveness of the system. For example, if a vehicle is traveling at 100 km/h, then a 10 ms delay corresponds to almost 30 cm of additional travel. In a braking or steering context, that extra distance can matter.

The main weakness of this class of protocol is therefore not only the added bandwidth. It is the loss of timing predictability. Automotive systems often require bounded worst-case behavior, not only good average performance. A security method that makes message delivery more variable can create a new safety problem while trying to solve a cybersecurity problem.

3.3.2 Payload Processor (Data Compression)

Another software-only approach is to compress the original CAN payload in order to create free space for security metadata. The basic idea is simple: if the application data can be represented using fewer bits, then the remaining bits can be used for a freshness value or authentication tag. Techniques such as Huffman coding or other compression methods can be applied to reduce the size of the data field.

This approach is appealing because it does not require changes to the CAN controller or transceiver. It also tries to preserve the single-frame structure of CAN communication. In theory, if the payload can be compressed reliably, then authentication information can be added without sending extra frames.

Critique - Non-Deterministic Execution: The difficulty is that compression is usually data-dependent. Some messages compress well, while others do not. For example, repeated or predictable values may be reduced significantly, but high-entropy or rapidly changing data may not shrink at all. This means the amount of free space created by compression is not always guaranteed.

There is also a timing problem. Compression and decompression may take different amounts of time depending on the data pattern. In an ordinary file-transfer system, this may be acceptable. In a safety-critical embedded system, it is more problematic. Automotive software must often be analyzed in terms of deterministic execution time. Engineers need to know the worst-case time required for a task, not only the average time.

For this reason, compression-based security can be difficult to certify or reason about in a real-time setting. The system may behave differently depending on the payload content. In contrast, $LCAP$ is designed so that the payload structure and verification

process remain fixed. The same type of computation is performed for every frame, which makes the timing behavior easier to analyze.

3.4 Gap Analysis and the LCAP Position

The literature shows that CAN security has no easy solution. Strong cryptographic methods often require more space, more computation, or more communication than Classical CAN can comfortably support. On the other hand, methods that are easy to deploy may provide weaker or less predictable security. This creates a gap between theoretical strength and practical usability.

Physical-layer methods try to avoid the 8-byte payload limit, but they may require modified hardware and can be sensitive to electrical noise. Hardware-assisted designs can provide strong security, but they are difficult to apply to the existing fleet of vehicles. Multi-frame software methods can use longer authentication values, but they increase bus load and may introduce jitter. Compression-based methods may create payload space, but their behavior depends on the data and may not be deterministic.

This gap is the space where *LCAP* is positioned. The aim is not to design the strongest possible cryptographic construction in isolation. The aim is to design a security mechanism that respects the real constraints of Classical CAN. In this thesis, the most important constraints are the 8-byte payload limit, the need for real-time behavior, the use of low-resource ECUs, and the requirement for backward compatibility.

LCAP is therefore designed as a deterministic, single-frame authentication protocol. It does not require new transceiver hardware. It does not split authentication data across multiple frames. It does not depend on compression to make room for security metadata. Instead, it divides the existing 64-bit CAN payload into application data, freshness information, and a truncated authentication tag.

This design involves trade-offs, especially in the size of the authentication tag. However, these trade-offs are made deliberately and are analyzed within the physical limits of the CAN bus. The protocol uses the fact that an attacker must operate online on a real bus, where every forgery attempt consumes time and creates observable traffic.

LCAP is positioned to fill this gap. We prioritize **Single-Frame Determinism** and **Resource Minimization** above all else, ensuring that security serves the system rather than slowing it down.

Chapter 4

The Formal Design of *LCAP*

The main contribution of this research is the design and formal definition of the Lightweight CAN Authentication Protocol (*LCAP*). The earlier chapters explained why Classical CAN is vulnerable and why many existing security mechanisms are difficult to apply in legacy vehicles. This chapter now presents the protocol itself in a more precise way. Instead of only describing the payload at a high level, we define how the 64-bit CAN data field is divided, how the authentication tag is produced, how freshness is handled, and how the receiver decides whether a frame should be accepted or rejected.

The design of *LCAP* is shaped by a practical limitation that cannot be ignored: Classical CAN gives only 8 bytes of data per frame. Because of this, the protocol cannot simply attach a large cryptographic tag or send extra metadata in the same way as conventional computer networks. The protocol must therefore balance three needs at the same time: keeping useful application data, preventing replay attacks, and adding authentication without increasing the number of CAN frames.

4.1 Single-Frame Design Philosophy

The design of *LCAP* follows a security-within-constraints approach. In this work, the 64-bit CAN payload is treated as a hard boundary rather than a flexible space. The protocol is not allowed to depend on extra frames, larger payloads, or modified CAN hardware. This is important because the purpose of *LCAP* is to support practical deployment on Classical CAN systems, including older ECUs with limited memory and processing power.

Mathematically, the 64-bit CAN payload is treated as a fixed boundary condition. All information required for security must fit inside this space. By placing the application data, freshness counter, and authentication tag into one 64-bit data field, the protocol preserves several important system properties:

- **Arbitration Invariance:** The protocol does not modify the CAN identifier field. This means that the normal priority-based bitwise arbitration mechanism of CAN is preserved. High-priority messages keep their original timing behavior, and the bus scheduling logic does not need to be redesigned.
- **Temporal Determinism:** Since each authenticated message remains a single CAN frame, the Worst-Case Execution Time (WCET) can be treated as a bounded and

predictable function of the hardware speed. The protocol avoids the variable delays that can appear when security metadata is spread across several frames.

- **Zero Fragmentation Overhead:** There is no need for a transport layer to split, track, and reassemble authentication fragments. This is especially useful for 8-bit controllers, where RAM and Flash memory are limited resources.

The single-frame design is therefore not only a formatting choice. It is a deliberate decision to protect the real-time behavior of the CAN bus. In an automotive control network, a security protocol must not make the system less predictable. For this reason, $LCAP$ prioritizes deterministic operation even though it requires careful use of the limited payload space.

4.2 Payload Partitioning and Entropy Allocation

The 64-bit CAN data field is a small but valuable resource. It must be divided in a way that keeps enough application data for the system to remain useful, while also reserving space for freshness and authentication. In $LCAP$, this allocation is treated as a security-utility trade-off. More bits for application data improve system usefulness, more bits for the counter improve replay resistance, and more bits for the authentication tag improve forge resistance.

The selected layout divides the payload into three fields: 24 bits for application data, 24 bits for the freshness counter, and 16 bits for the authentication tag. This gives the following total:

$$24 + 24 + 16 = 64.$$

This structure allows one complete authenticated message to fit inside a single Classical CAN frame.

LCAP single-frame security layout inside one Classical CAN payload

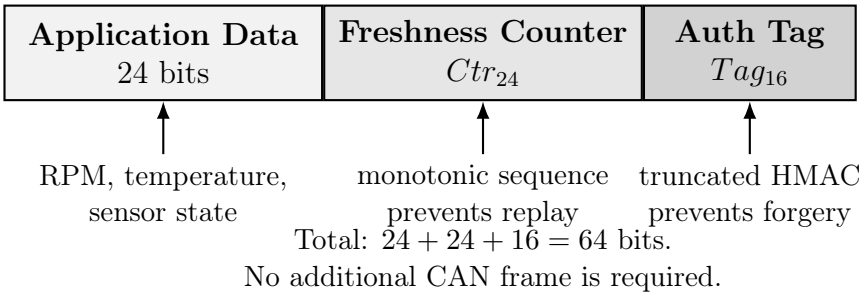


Figure 4.1: LCAP payload partitioning. The 64-bit Classical CAN data field is divided into application data, a freshness counter, and a truncated authentication tag.

The bit-level mapping of the $LCAP$ frame is defined in the following table:

Field	Size (Bits)	Mathematical Role	Data Type
Application Data	24	System State ($\mathcal{D}$)	Binary Vector
Freshness (Ctr_{24})	24	Monotonic Sequence ($\mathcal{C}$)	Unsigned Int
Auth Tag (Tag_{16})	16	Message Integrity Code ($\mathcal{T}$)	Truncated Hash

Table 4.1: The LCAP 64-bit Payload Allocation Map.

4.2.1 Application Data Partitioning ($\mathcal{D}$)

Although $LCAP$ is designed to be data-agnostic, the experimental prototype needs a concrete data format for validation. In this implementation, 24 bits are reserved for application-layer signals. These 24 bits are divided into a 16-bit RPM value and an 8-bit temperature value. This is a useful test case because it represents the type of compact numerical data commonly found in automotive communication.

This allocation also demonstrates an important point: adding authentication does not require removing all useful application information from the CAN frame. The payload still carries meaningful sensor data, while also including freshness and authentication fields. In other words, $LCAP$ does not turn the CAN frame into a pure security message. It keeps the frame useful for the control system while adding a security layer around it.

4.2.2 Freshness Counter Analysis ($\mathcal{C}$): Temporal Longevity

A 24-bit counter (Ctr_{24}) was selected to provide a practical balance between payload cost and replay protection. The counter is large enough to support long operating periods, but small enough to leave room for both application data and the authentication tag.

Counter Overflow and Epochs: Given a periodic transmission frequency f measured in Hz, the time T_{wrap} required for the counter to overflow and return to zero is:

$$T_{wrap} = \frac{2^{24}}{f} \text{ seconds} \tag{4.1}$$

For a high-priority safety signal transmitting at a frequency of 100 Hz, meaning one frame every 10 ms, the wraparound time is:

$$T_{wrap} = \frac{16,777,216}{100} = 167,772.16 \text{ seconds} \approx 46.6 \text{ hours}$$

This means that even under a relatively fast periodic transmission rate, the counter can run for almost two continuous days before wrapping around. For most legacy vehicles, a continuous driving epoch of 46.6 hours is more than sufficient. In a practical implementation, counter overflow would not be ignored. It would trigger a key rotation, epoch synchronization, or reinitialization event. This issue is discussed again in Chapter 9 as part of the future work and deployment considerations.

4.3 Cryptographic Transformation and Entropy Reduction

The authentication mechanism in $LCAP$ is based on symmetric-key cryptography. Sender and receiver share a secret key $K \in \{0,1\}^{128}$. The sender uses this key to compute an authentication tag, and the receiver uses the same key to recompute the expected tag. If the two tags match and the counter is fresh, the frame is accepted.

A symmetric-key design was chosen because it is realistic for low-resource ECUs. Public-key cryptography provides useful properties in many systems, but it is too expensive for the timing and memory constraints considered in this thesis. In contrast, a symmetric construction such as HMAC can be implemented on 8-bit hardware with careful optimization.

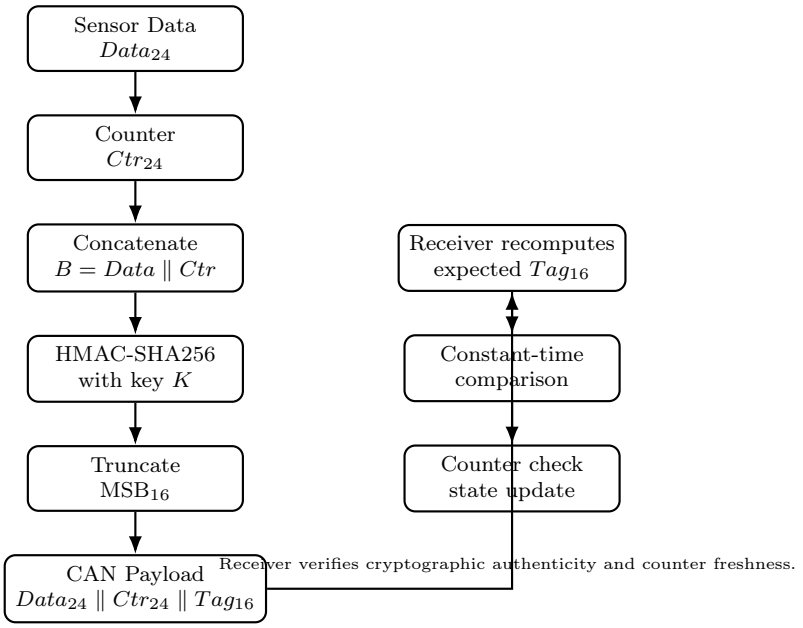


Figure 4.2: LCAP authentication pipeline. The sender computes a truncated HMAC over application data and freshness information, while the receiver recomputes the tag and validates the counter before accepting the frame.

4.3.1 Concatenation and Input Construction

The input to the cryptographic function is a bit string B formed by joining the current application data and the freshness counter:

$$B = Data_{24} \parallel Ctr_{24} \tag{4.2}$$

This construction is important because the counter changes with every transmission. Even if the application data remains unchanged, the HMAC input still changes. For example, if the car is idling and the RPM value remains constant, the counter continues to increase. As a result, the authentication tag also changes from frame to frame.

This property is essential for replay protection. An attacker may observe a valid frame on the bus, but replaying the same frame later will not be accepted once the receiver has moved to a newer counter value. The tag authenticates the data and counter together, so the receiver is not only checking whether the data is valid, but also whether the data belongs to the correct point in the message sequence.

4.3.2 The Truncation Function τ

The full output of HMAC-SHA256 is 256 bits. Classical CAN cannot carry such a large tag together with useful application data and a freshness counter. For this reason, $LCAP$ applies a truncation function τ and keeps only the most significant 16 bits:

$$Tag_{16} = \mathrm{MSB}_{16}(HMAC_{SHA256}(K, B)) \tag{4.3}$$

From a pure cryptographic perspective, reducing a 256-bit value to 16 bits is a major loss of entropy. In a conventional internet-facing system, a 16-bit authentication tag would be considered too short. However, the operating environment of Classical CAN is different. The attacker is not allowed to test guesses offline at unlimited speed. Instead, each forgery attempt must be transmitted as a physical CAN frame on the bus.

This distinction matters. In an automotive CAN network, the baud rate, arbitration process, and bus utilization limit the number of guesses an attacker can make in a given time. Therefore, the security of the truncated tag must be evaluated together with the physical and timing constraints of the network. As shown later in Chapter 8, the physical CAN bus acts as a natural rate limiter against brute-force attempts.

4.4 Receiver State Machine

The receiver in $LCAP$ does more than compare two tag values. It also maintains state. This is necessary because freshness cannot be verified from a single frame alone. The receiver must remember the last accepted counter and decide whether the newly received counter is valid, stale, or unexpectedly far ahead.

We formally define the receiver state machine as a 5-tuple $(S, \Sigma, \delta, s_0, F)$:

- $S =$ { IDLE, VERIFYING, ACCEPT, SOFT_RESYNC, REJECT_STALE, REJECT_BAD_TAG }
- $\Sigma =$ { Incoming CAN Frame }
- $s_0 =$ IDLE
- $\delta : S \times \Sigma \to S$ (The transition function)

In practical terms, the receiver waits in the `IDLE` state until a frame arrives. It then moves to `VERIFYING`, recomputes the expected tag, checks the counter, and selects the appropriate result state. After handling the frame, it returns to `IDLE` and waits for the next message.

4.4.1 Decision Logic and State Transitions

The decision made in the `VERIFYING` state depends on two values. The first is the tag validity variable $\mathcal{V}$, where $\mathcal{V} = 1$ means that the received tag matches the locally recomputed tag, and $\mathcal{V} = 0$ means that it does not. The second is the counter difference:

$$\Delta = C_{new} - C_{last}.$$

The receiver uses these values to choose one of the following transitions:

1. **Transition to ACCEPT:** If $\mathcal{V} = 1$ and $1 \leq \Delta \leq W$. This is the normal operating case. The tag is valid, and the counter has moved forward within the permitted window.

2. **Transition to SOFT_RESYNC:** If $\mathcal{V} = 1$ and $\Delta > W$. This means the frame is authentic, but the counter has jumped farther than expected. This may happen if many frames were lost or if the receiver was temporarily unavailable.

3. **Transition to REJECT_STALE:** If $\mathcal{V} = 1$ and $\Delta \leq 0$. This means the frame may be cryptographically valid, but it is not fresh. It is therefore treated as a replay or stale message.

4. **Transition to REJECT_BAD_TAG:** If $\mathcal{V} = 0$. This means the frame failed authentication. It may be a forged message, a corrupted frame, or a frame generated with the wrong key.

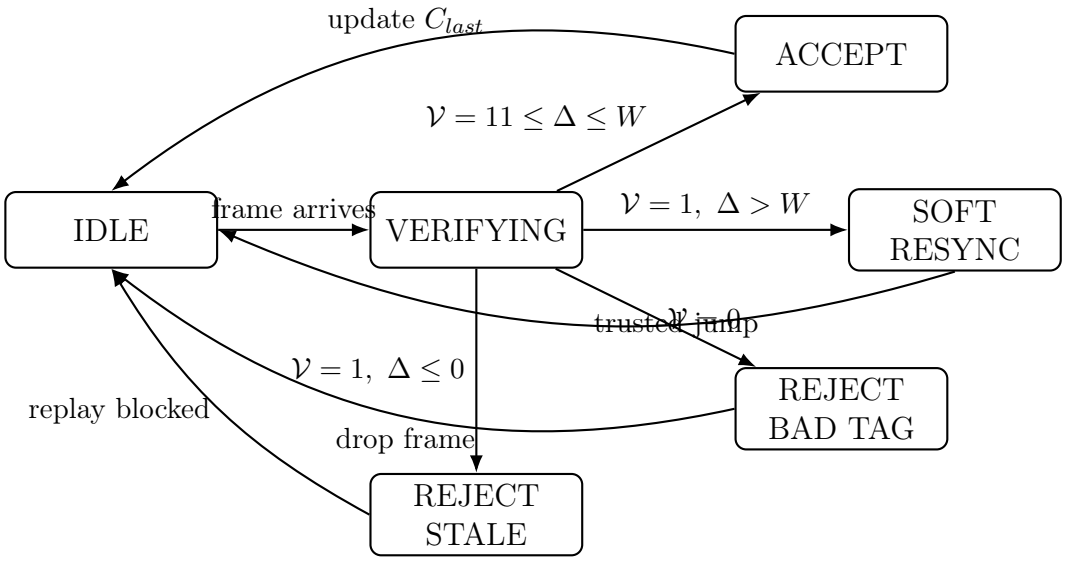
4.5 The Soft-Resynchronization Mechanism: Handling Noise

One of the practical features of $LCAP$ is its handling of ordinary frame loss. A real vehicle is not an ideal communication environment. Electrical noise, electromagnetic interference, temporary bus contention, and controller-level errors can all cause occasional message loss. A security protocol that treats every missed frame as an attack would be too fragile for this setting.

A strict counter rule would require the receiver to accept only frames where $\Delta = 1$. This would be secure against replay, but it would also be too rigid. If one legitimate frame is lost because of noise, then the next valid frame would have $\Delta = 2$ and would be rejected. This could turn a small communication error into a persistent denial-of-service condition.

To avoid this, $LCAP$ uses a sliding acceptance window W. In the prototype and analysis, this value is set to $W = 16$. This means that the receiver can tolerate a limited number of missed frames. For example, if five frames are lost, the sixth frame can still be accepted because the tag is valid and the counter jump is still inside the permitted window.

The important point is that soft resynchronization does not remove authentication. The receiver does not accept a large counter value only because it is larger than the



$\mathcal{V}$ = tag validity, $\Delta = C_{new} - C_{last}$, W = soft-resynchronization window.

Figure 4.3: Finite state machine of the LCAP receiver. A frame is accepted only when both the authentication tag and freshness counter satisfy the protocol logic.

previous one. The tag must still be valid. Therefore, an attacker cannot simply force the receiver forward by guessing a future counter value unless they can also produce the correct authentication tag.

4.6 Determinism Worst-Case Execution Time (WCET)

In safety-critical systems, average execution time is not enough. What matters most is whether the system can always complete its task before the deadline. This is why Worst-Case Execution Time (WCET) is central to automotive software analysis and safety certification.

$LCAP$ is designed to be deterministic. The protocol always processes the same basic payload structure, applies the same HMAC computation, performs the same truncation, and checks the same counter logic. This makes the execution path predictable. The computation does not depend on whether the application data happens to be compressible or whether additional authentication fragments arrive on time.

This is different from compression-based approaches, where processing time and payload size can vary depending on the data. It is also different from multi-frame authentication approaches, where the receiver may need to wait for several related frames before it can make a decision. In those cases, timing behavior depends not only on the ECU but also on bus load and arbitration delays.

By keeping authentication inside one frame, $LCAP$ allows engineers to reason about the security layer as a bounded operation. The receiver can verify a frame, update its state, and return to normal operation without waiting for extra fragments or handshakes. This

supports the main design goal of the protocol: to add authentication without damaging the timing predictability of Classical CAN.

Chapter 5

Implementation: Hardware Schematics and Software Stack

After defining the mathematical structure of *LCAP*, the next step is to show that the protocol can actually run on real embedded hardware. A protocol may look correct on paper, but in an automotive environment it must also respect timing limits, memory limits, hardware interfaces, and the behavior of the CAN controller itself. For this reason, the implementation stage is an essential part of validating the proposed design.

This chapter describes the hardware-in-the-loop testbed used to implement and evaluate *LCAP*. The discussion moves from the abstract state machines and protocol definitions introduced earlier into the practical details of embedded systems engineering. It explains the selected hardware components, the communication interfaces between them, and the software structure used to support authentication and verification. By implementing *LCAP* on 8-bit legacy hardware, this work shows that the protocol is not limited to modern high-performance ECUs, but can also operate on resource-constrained platforms that are closer to older automotive systems.

5.1 The Hardware Testbed Philosophy: Designing for the Computational Lower Bound

The hardware platform was selected with a conservative design philosophy. Instead of beginning with a powerful automotive processor, the implementation was built on low-cost 8-bit microcontrollers. This choice was intentional. Modern vehicles increasingly use 32-bit ARM Cortex-R, Infineon AURIX, and other high-performance embedded processors, but the existing global vehicle fleet contains a much wider range of hardware. Many older ECUs have limited clock speed, small memory capacity, and no built-in cryptographic acceleration.

For this reason, the prototype was designed around a "Worst-Case Scenario" approach. If *LCAP* can run within acceptable timing limits on an 8-bit microcontroller, then the same protocol should run with greater ease on more capable automotive hardware. In this sense, the testbed establishes a practical lower bound for performance.

We therefore implemented *LCAP* on 8-bit microcontrollers to evaluate whether the

most expensive parts of the protocol, especially the HMAC-SHA256 calculation and the receiver-side sliding-window state machine, can operate under realistic timing constraints. This approach supports the backward-compatibility goal of the thesis. The aim is not only to secure future vehicles, but also to show that a lightweight authentication method can be applied to older embedded platforms without requiring a complete hardware redesign.

Detailed Node Specifications and Hardware Architecture

Each node in the distributed testbed, whether acting as sender or receiver, is built from three main layers. The first is the Processing Layer, where the microcontroller executes the *LCAP* logic. The second is the Protocol Controller Layer, where the CAN controller handles frame-level communication. The third is the Physical Interface Layer, where the transceiver drives the differential CAN bus signals.

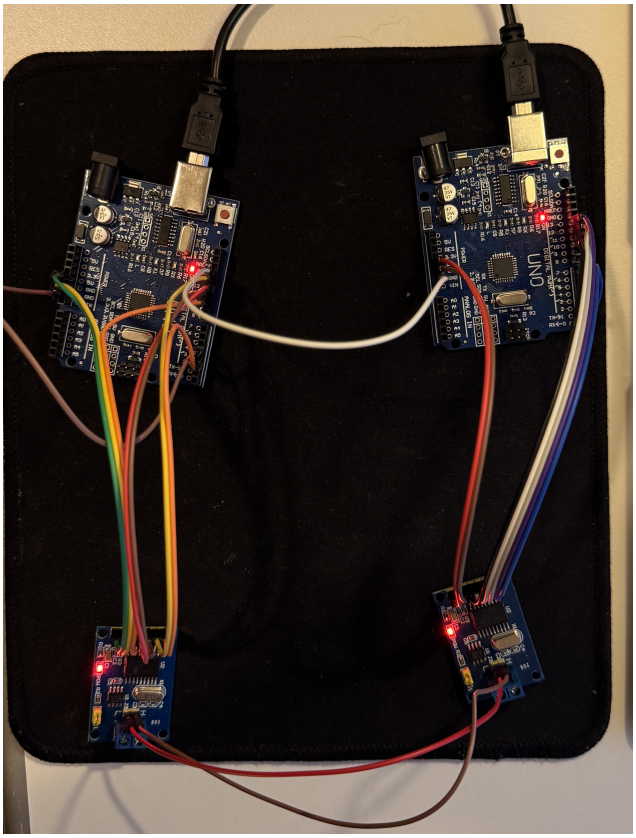


Figure 5.1: Physical hardware-in-the-loop LCAP testbed. The setup consists of two Arduino Uno boards connected to MCP2515 CAN controller modules and TJA1050 transceivers, forming a terminated Classical CAN network for sender–receiver authentication experiments.

5.1.1 Physical Prototype Description

Figure 5.1 shows the physical prototype used during the validation of the proposed *LCAP* design. The setup consists of two Arduino Uno boards, and each board is connected to an MCP2515 CAN controller module. The MCP2515 modules provide CAN 2.0B

communication through the SPI interface, while the Arduino boards execute the sender-side and receiver-side protocol logic.

The left node operates as the sender ECU. It prepares the application data, increments the freshness counter, computes the truncated authentication tag, and transmits the final 8-byte CAN payload. The right node operates as the receiver ECU. It listens for the expected CAN traffic, extracts the received payload, recomputes the authentication tag, checks the freshness counter, and then accepts or rejects the frame according to the receiver state machine.

The physical testbed is deliberately simple. This simplicity is useful because the goal of the thesis is not to demonstrate the protocol on expensive or high-end automotive processors. Instead, the aim is to show that authentication is possible even on low-cost 8-bit microcontrollers. For this reason, the Arduino Uno platform serves as a conservative lower-bound approximation for legacy embedded ECUs.

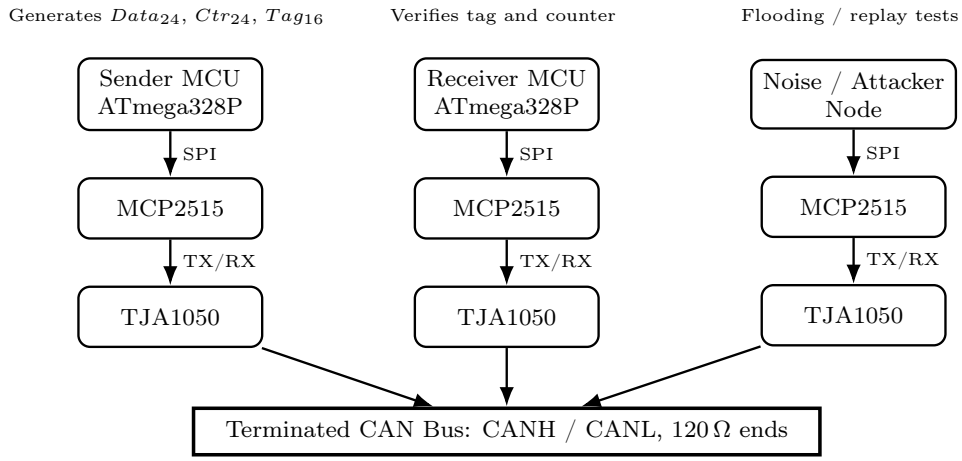


Figure 5.2: Hardware-in-the-loop validation architecture for LCAP. The testbed contains a sender node, receiver node, and adversarial/noise node connected through a terminated Classical CAN bus.

5.1.2 Processing Layer: The ATmega328P Microcontroller

The processing core of each testbed node is the ATmega328P, which is an 8-bit AVR RISC-based microcontroller. Although it is modest compared with modern automotive processors, it is useful for this research because it forces the implementation to respect strict resource limits. If the protocol can run here, then it provides strong evidence that the design is lightweight enough for legacy embedded systems.

For this project, the ATmega328P presents several important constraints:

- **Register Architecture:** The ATmega328P uses 8-bit registers. This is a major limitation for SHA-256, because SHA-256 is naturally designed around 32-bit word operations. As a result, each 32-bit addition, shift, or rotation must be broken into several smaller 8-bit operations. This increases the instruction count and creates a practical 4× penalty when compared with a native 32-bit architecture.

- **Clock Speed:** The microcontroller runs at 16 MHz using an external crystal oscillator. This gives a cycle period of 62.5 nanoseconds. This timing resolution is useful when reasoning about processing time, latency, and jitter during authentication.
- **Flash Memory:** The platform provides 32 KB of Flash memory. The *LCAP* logic together with the SHA-256 implementation occupies a meaningful portion of this space, but still leaves room for the remaining ECU application logic. This is important because an authentication layer must not consume the entire memory budget of a legacy controller.
- **SRAM:** The ATmega328P provides only 2 KB of SRAM. This is one of the strictest limits in the prototype. Cryptographic operations require temporary buffers, especially for SHA-256 message scheduling and HMAC processing. For this reason, memory must be handled carefully, and dynamic allocation must be avoided.

5.1.3 Protocol Layer: The MCP2515 Standalone CAN Controller

The ATmega328P does not include a native CAN peripheral. Because of this, the prototype uses the MCP2515 standalone CAN controller. The MCP2515 communicates with the microcontroller over SPI and handles the CAN frame-level operations in hardware. This separation is useful because the microcontroller can focus on the *LCAP* logic, while the MCP2515 manages the details of CAN communication.

The MCP2515 implements CAN 2.0B behavior and is responsible for several important functions:

- **Bitwise Arbitration:** The controller handles the CAN arbitration process, including the dominant and recessive bit behavior described in Chapter 2. This ensures that the prototype follows the normal priority rules of a Classical CAN network.
- **CRC Generation:** The controller appends the standard 15-bit CAN CRC field to transmitted frames. This CRC is useful for detecting accidental transmission errors, although it is not a cryptographic integrity check.
- **Message Buffering:** The MCP2515 provides transmit and receive buffers. These buffers help decouple the timing of CAN communication from the timing of microcontroller processing. This is useful when the MCU is busy verifying a received frame or preparing the next authenticated payload.

5.1.4 Physical Layer: The TJA1050 High-Speed CAN Transceiver

The TJA1050 transceiver provides the physical connection to the CAN bus. The MCP2515 and ATmega328P operate using digital logic-level signals, but the CAN bus itself uses differential signaling over CANH and CANL. The transceiver performs this conversion.

In transmission, the TJA1050 converts the controller's digital TX signal into the corresponding differential voltages on the bus. In reception, it converts the CANH and CANL voltage difference back into a digital RX signal that the CAN controller can process. This physical layer is essential because it allows the prototype to behave like a real Classical CAN node.

The transceiver also helps support electromagnetic compatibility. Automotive environments can contain noise from ignition systems, motors, switching electronics, and wiring harnesses. While the prototype is a laboratory setup, using a proper CAN transceiver makes the testbed closer to a realistic in-vehicle communication structure.

5.2 SPI Protocol and Timing Budget

The ATmega328P and MCP2515 communicate through the **Serial Peripheral Interface (SPI)**. This interface is important because every transmitted or received CAN frame must pass through it. Therefore, SPI communication contributes to the total latency of the system.

Although SPI is much faster than the CAN bus itself, it still has overhead. The microcontroller must issue commands, transfer addresses and data, handle interrupts or polling logic, and process the result. These steps are small individually, but they matter when the goal is to evaluate real-time behavior.

5.2.1 SPI Configuration

In the prototype, the SPI bus was configured to operate at 10 MHz, which is the maximum stable frequency used for this setup. The timing budget for one 8-byte transfer can be modeled as:

$$T_{transfer} = \frac{N_{bits}}{F_{SPI}} + T_{overhead} \tag{5.1}$$

Here, N_{bits} represents the number of bits transferred over SPI. This includes the 64-bit CAN payload as well as the command and address information needed by the MCP2515. With an 8-byte payload and a 16-bit command/address overhead, the raw transfer time at 10 MHz is approximately $8\,\mu$s.

However, this value only represents the raw bit transfer time. In practice, additional time is introduced by software control, interrupt handling, register access, and context switching. This extra component is represented by $T_{overhead}$. The effect of this overhead is examined later in the results chapter, where the total authentication and verification latency is evaluated.

5.3 Software Architecture and Memory Strategy

The *LCAP* software stack was implemented in C++. The choice of C++ allows direct hardware access while still supporting structured code organization. Since the target environment is safety-critical, the implementation follows the spirit of **MISRA C** principles, especially with respect to predictable control flow, clear memory usage, and avoiding unsafe dynamic behavior.

The software is divided into two main parts. The first part handles CAN initialization, frame transmission, frame reception, and role selection. The second part contains the *LCAP* logic, including payload packing, HMAC-SHA256 tag generation, tag truncation, and receiver verification.

5.3.1 Deterministic Memory Allocation

Dynamic memory allocation is avoided in the implementation. In automotive real-time systems, the use of `malloc()` or `free()` is generally unsafe because allocation time can vary and heap fragmentation can occur over long execution periods. On a small microcontroller with only 2 KB of SRAM, these risks become even more serious.

For this reason, all buffers required by *LCAP* are allocated statically or on the stack with fixed sizes. This includes the buffers used for the SHA-256 message block, HMAC inner and outer padding, and CAN payload construction. The advantage of this approach is that memory usage remains predictable throughout execution.

This deterministic memory strategy is important for two reasons. First, it reduces the risk of runtime memory failure. Second, it makes the protocol easier to analyze from a real-time perspective. Since the memory footprint does not grow unpredictably during operation, the software behavior is more suitable for embedded control environments.

5.3.2 Cryptographic Library Optimization

Most SHA-256 implementations are written with 32-bit or 64-bit processors in mind. On an 8-bit AVR processor, these implementations are not ideal because operations on 32-bit words require multiple 8-bit instructions. For this prototype, a compact byte-oriented implementation was used so that the algorithm could run within the limits of the ATmega328P.

The implementation minimizes unnecessary copying and avoids dynamic buffers. The message construction is kept small because *LCAP* authenticates only the 48-bit combination of application data and freshness counter. The resulting HMAC-SHA256 output is then truncated to 16 bits before being packed into the final CAN payload.

A Big-Endian mapping is used for the payload fields. This keeps the transmitted byte order clear and consistent with the way the CAN payload is parsed on the receiver side. It also reduces unnecessary byte rearrangement during packing and unpacking.

5.4 Algorithmic Logic Flow and State Management

The software behavior of *LCAP* can be understood through two routines: the sender routine and the receiver routine. The sender builds authenticated frames, while the receiver validates them. Both routines are designed to be simple and predictable, which is important for real-time operation.

5.4.1 The Sender Routine (The "Packer")

The sender routine prepares the 8-byte authenticated CAN payload. It is intended to run periodically, without interfering with the main ECU task. For every transmission cycle, the sender performs the following operations:

1. **Sampling:** The 24 bits of application data are captured. In the prototype, this consists of a 16-bit RPM value and an 8-bit temperature value.

2. **Freshness:** The 24-bit counter is updated. In a production system, the counter state may be initialized from persistent memory at boot and then maintained in SRAM during operation.

3. **Hashing:** HMAC-SHA256 is computed over the 48-bit block formed by the application data and the freshness counter.

4. **Truncation:** The 16 most significant bits of the HMAC output are selected and inserted into the final 8-byte CAN frame.

After these steps, the frame is sent through the MCP2515 CAN controller. The final payload contains the application data, the freshness counter, and the authentication tag. This allows the receiver to verify both integrity and freshness without requiring any additional CAN frame.

5.4.2 The Receiver Routine (The "Validator")

The receiver routine is responsible for checking whether an incoming CAN frame is valid. In the prototype, the receiver can use the MCP2515 interrupt line to avoid continuous polling. The interrupt signal indicates that a frame is available, allowing the microcontroller to process the message only when necessary.

The receiver uses an **Active-Low External Interrupt** (INT0). This helps reduce wasted CPU cycles because the ECU does not need to constantly check the bus in a tight loop. Instead, it reacts when a message is ready for verification.

The receiver performs the following steps:

1. **Interrupt Handling:** The MCU responds to the receive event and retrieves the 8-byte payload from the MCP2515 using SPI.

2. **Parallel Verification:** The payload is unpacked into application data, counter, and received tag. The MCU then recomputes the expected tag from the received data and counter.

3. **Constant-Time Comparison:** To reduce timing side-channel leakage, the received $Tag16$ is compared with the recomputed tag using a fixed comparison procedure rather than an early-exit comparison.

4. **Logic Resolution:** Based on the tag result and counter freshness, the receiver updates its state as ACCEPT, REJECT, or SOFT_RESYNC.

This receiver-side logic is important because authentication alone is not enough. A replayed message may still have a valid tag, since it was originally produced by the legitimate sender. Therefore, the receiver must also check the freshness counter before accepting the frame.

5.5 Network Topology and Testbed Scenarios

The testbed simulates a small ISO 11898-style Classical CAN network. The network is intentionally simple so that the behavior of *LCAP* can be observed clearly. The setup contains sender and receiver nodes, and it can also include an additional node used to generate disturbance or adversarial traffic.

- **Termination:** Two 120 Ω termination resistors are used at the ends of the CAN bus. This matches the characteristic impedance of the bus and helps prevent signal reflections that could cause communication errors.
- **Scenario A (Nominal):** In the nominal case, the sender transmits authenticated frames at the configured rate, and the receiver verifies them under normal conditions. This scenario is used to confirm that the basic authentication and freshness logic works correctly.
- **Scenario B (Adversarial):** In the adversarial case, an additional node is introduced. This node can inject forged or disruptive traffic using the MCP2515. The purpose of this scenario is to observe how the receiver reacts to invalid tags, missed frames, and possible replay-like behavior. It is also useful for evaluating the practical value of the Adaptive Rate Limiting policy.

Together, these scenarios provide a practical foundation for the experimental evaluation. The nominal case verifies that *LCAP* works correctly during normal communication, while the adversarial case examines how the protocol behaves when the CAN bus is disturbed or attacked.

Chapter 6

Experimental Methodology and Test Scenarios

This chapter explains how the proposed Lightweight CAN Authentication Protocol ($LCAP$) was tested in practice. The previous chapter described the hardware components and the software stack used in the prototype. Here, the focus shifts to the evaluation process itself: how the tests were carried out, what assumptions were made, and how the different experimental scenarios were organized.

The purpose of this chapter is to connect the implementation work with the results that follow. For an automotive Cyber Physical System, a security protocol cannot be judged only by its mathematical design. It also has to work under realistic operating conditions, where messages are periodic, hardware resources are limited, and communication errors may occur. For this reason, the evaluation of $LCAP$ was built around three main questions:

1. Can the sender generate authenticated CAN frames within a real-time transmission interval?

2. Can the receiver verify authenticity and freshness without introducing unacceptable delay?

3. Can the protocol recover from ordinary communication disturbances without requiring expensive resynchronization messages?

The experiments were performed on a hardware-in-the-loop prototype using low-cost 8-bit microcontrollers. This choice was made deliberately. If the protocol can operate on a constrained platform such as the ATmega328P, then it is reasonable to expect that the same design would perform even better on modern 32-bit automotive microcontrollers.

6.1 Experimental Setup Overview

The experimental setup consists of a small Classical CAN network made up of sender and receiver nodes. Each node is built using an Arduino Uno board, an MCP2515 CAN

controller module, and a TJA1050 CAN transceiver. The Arduino Uno acts as the processing unit, the MCP2515 handles CAN 2.0B frame communication, and the TJA1050 provides the differential physical signaling required by the CAN bus.

The sender node represents a simplified Electronic Control Unit (ECU) that periodically transmits sensor information. In the prototype, the application data is simulated using fixed values for engine revolutions per minute and temperature. Even though these values are simulated, their structure is close to the type of compact numerical signals commonly found in automotive CAN messages.

The receiver node represents another ECU that consumes this information. However, instead of accepting every frame with the correct CAN identifier, the receiver applies the *LCAP* verification logic. It extracts the application data, freshness counter, and authentication tag from the received 8-byte payload. It then recomputes the expected authentication tag using the shared secret key and checks whether the received counter satisfies the freshness condition.

The testbed is intentionally small and controlled. This is useful because it isolates the behavior of the authentication protocol itself. In a complete vehicle network, there may be many ECUs, many message identifiers, and different priority levels. Still, the central problem remains the same: the receiver must decide whether an incoming CAN frame is fresh, authentic, and safe to process.

6.2 Sender and Receiver Roles

The sender and receiver have clearly separated responsibilities in the *LCAP* prototype. This separation makes it easier to observe the behavior of the protocol during testing and to identify which part of the system is responsible for each operation.

6.2.1 Sender Node

The sender node is responsible for constructing authenticated CAN frames. During each transmission cycle, it performs the following operations:

1. It prepares the application data field.
2. It increments the 24-bit freshness counter.
3. It concatenates the application data and counter.
4. It computes an HMAC-SHA256 value using the shared secret key.
5. It truncates the output to obtain a 16-bit authentication tag.
6. It packs the application data, counter, and tag into one 8-byte CAN payload.
7. It transmits the final frame using the MCP2515 CAN controller.

The sender therefore acts as the source of authenticated data. Every transmitted frame contains both the application information and the cryptographic evidence needed by the receiver to verify that the frame was produced by a legitimate node.

The main requirement for the sender is predictable timing. The sender must finish the cryptographic computation and transmit the CAN frame within the selected message period. In the prototype, the sender transmits periodically, which allows the timing behavior of the authentication process to be observed over many consecutive frames.

6.2.2 Receiver Node

The receiver node is responsible for validating incoming frames. It cannot trust a CAN frame only because the frame has the expected identifier. This is an important point, because Classical CAN identifiers describe the type and priority of a message, not the identity of the sender. A malicious or compromised node can transmit a frame using the same identifier as a legitimate ECU.

For each received frame, the receiver performs the following operations:

1. It reads the 8-byte CAN payload from the MCP2515.
2. It separates the payload into application data, counter, and received tag.
3. It recomputes the expected tag using the same shared secret key.
4. It compares the received tag with the recomputed tag.
5. It calculates the counter difference $\Delta = C_{new} - C_{last}$.
6. It updates its internal state according to the receiver state machine.

The receiver can produce several possible outcomes. A valid frame with an acceptable counter increment is accepted. A frame with a stale counter is rejected as a replay attempt. A frame with an invalid tag is rejected as a possible forgery or corrupted transmission. A frame with a valid tag but a larger counter jump may trigger soft resynchronization, depending on the size of the jump and the configured acceptance policy.

6.3 CAN Message Format Used in Testing

The experimental frame format follows the formal $LCAP$ payload structure defined earlier in the thesis. The complete CAN payload has a length of 8 bytes, which is the maximum payload size available in Classical CAN.

The payload is divided into three fields:

$$Payload_{64} = Data_{24} \parallel Ctr_{24} \parallel Tag_{16} \tag{6.1}$$

The first 24 bits contain application data. In the prototype, this field is divided into a 16-bit RPM value and an 8-bit temperature value. The next 24 bits contain the freshness counter. The final 16 bits contain the truncated authentication tag.

Payload Field	Size	Purpose
Application Data	24 bits	Carries simulated RPM and temperature values
Freshness Counter	24 bits	Prevents replay by enforcing monotonic progression
Authentication Tag	16 bits	Provides message authenticity and integrity

Table 6.1: LCAP payload format used during experimental testing.

The CAN identifier used in the prototype is `0x500`. This identifier was chosen as a stable experimental message ID so that the sender and receiver could coordinate during testing. The security of *LCAP* does not depend on keeping this identifier secret. Instead, security comes from the shared secret key used in the HMAC computation and from the freshness counter used to prevent replay.

The sender output logs show that the authentication tag changes from frame to frame even when the application data remains constant. This happens because the freshness counter is included in the HMAC input. Therefore, the input to the cryptographic function changes on every transmission, even if the physical sensor value itself does not change.

6.4 Test Scenario A: Nominal Authentication

The first scenario evaluates the normal operating condition of the protocol. In this scenario, the sender periodically transmits valid authenticated frames, and the receiver checks each frame using the expected key and counter logic.

The objective of this scenario is to confirm that the protocol works correctly when there is no attack and no intentional communication disturbance. This is the baseline case. If the protocol does not behave correctly under normal conditions, then more complex attack or recovery experiments would not be meaningful.

During nominal testing, the following behavior is expected:

- The sender increments the counter for every transmitted frame.
- The authentication tag changes as the counter changes.
- The receiver recomputes the same tag as the sender.
- The receiver accepts frames whose counters are newer than the previous accepted counter.
- No resynchronization or rejection state is triggered.

This scenario verifies the basic correctness of the cryptographic transformation and the payload packing logic. It also verifies that the receiver can parse the received 8-byte frame correctly and reconstruct the same HMAC input used by the sender.

Parameter	**Value Used in Scenario A**
CAN identifier	`0x500`
Application data	Simulated RPM and temperature
Counter behavior	Incremented once per frame
Expected receiver output	`ACCEPT`
Attack traffic	None
Purpose	Validate normal authentication behavior

Table 6.2: Configuration of the nominal authentication test scenario.

The nominal scenario is also useful for observing how the authentication tag varies. A secure message authentication code should appear unpredictable to an attacker who does not know the key. Therefore, small changes in the counter should produce authentication tags that appear unrelated to one another.

6.5 Test Scenario B: Transmission Failure and Recovery

The second scenario evaluates how $LCAP$ behaves when communication is interrupted or frames are lost. Frame loss can happen in real automotive networks because of electromagnetic interference, bus contention, wiring problems, or temporary controller errors. A practical security protocol must therefore tolerate some communication imperfection.

In strict counter-based protocols, the receiver may reject every future frame after a single lost message if it expects the next counter value to be exactly $C_{last} + 1$. This behavior can be dangerous in an automotive setting because ordinary electrical noise could turn into a denial-of-service condition. $LCAP$ avoids this problem by using a soft-resynchronization window.

The receiver accepts a valid authenticated frame if the counter jump is within the acceptable range:

$$1 \leq \Delta \leq W \tag{6.2}$$

where W is the soft-resynchronization window. In the prototype, the window is set to:

$$W = 16 \tag{6.3}$$

This means that the receiver can tolerate a limited number of missed frames. If several frames are lost, but the next received frame has a valid authentication tag and a counter value inside the permitted window, the receiver can safely update its state and continue operation.

The purpose of Scenario B is to verify that the protocol can recover without requiring an additional handshake. This matters because extra resynchronization messages would increase bus load and could interfere with real-time control traffic.

Parameter	**Value Used in Scenario B**
Failure type	Temporary transmission interruption or frame loss
Receiver mechanism	Soft-resynchronization window
Window size	$W = 16$
Expected result	Receiver accepts next valid frame within window
Extra CAN frames required	None
Purpose	Evaluate recovery after missed frames

Table 6.3: Configuration of the transmission failure and recovery scenario.

The important security point is that the receiver does not accept a counter jump simply because the counter is larger. The authentication tag must also be valid. Therefore, an attacker cannot choose an arbitrary future counter value and force the receiver to resynchronize. Without the secret key, the probability of guessing the correct 16-bit tag remains limited by the truncated tag space and by the physical injection rate of the CAN bus.

6.6 Test Scenario C: Replay and Stale Counter Rejection

The third scenario evaluates replay resistance. Replay attacks are one of the most important threats against Classical CAN because standard CAN frames do not include timestamps, counters, nonces, or sender authentication. An attacker can record a valid frame and retransmit it later, trying to force the receiver back into an old state.

In $LCAP$, replay attacks are handled by the monotonic freshness counter. The receiver stores the most recently accepted counter value, denoted by C_{last}. When a new frame arrives, the receiver computes:

$$\Delta = C_{new} - C_{last} \tag{6.4}$$

If the value of Δ is less than or equal to zero, the frame is stale:

$$\Delta \leq 0 \tag{6.5}$$

A stale frame may still contain a valid authentication tag because it was originally generated by the legitimate sender. However, it must not be accepted because it does not represent a fresh message. For that reason, the receiver rejects it with the logical state `REJECT_STALE`.

This distinction is important. A valid tag proves that the frame was authentic when it was created. It does not prove that the frame is fresh at the time it is received. Freshness must therefore be checked separately using the counter.

Condition	Receiver Decision
Valid tag and $1 \leq \Delta \leq W$	`ACCEPT`
Valid tag and $\Delta > W$	`SOFT_RESYNC` or policy-dependent recovery
Valid tag and $\Delta \leq 0$	`REJECT_STALE`
Invalid tag	`REJECT_BAD_TAG`
Repeated invalid tags	Possible rate-limiting or lockdown response

Table 6.4: Receiver decisions used during freshness and authentication testing.

Scenario C shows why cryptographic integrity and counter-based freshness must be used together. Authentication by itself is not enough for a control network, because old commands can still be dangerous. A replayed brake, throttle, or door-lock message may cause harm even if it was originally produced by a legitimate ECU. For this reason, *LCAP* treats freshness as a core part of the authentication decision.

6.7 Measurement Procedure

The experimental measurements were collected by observing both sender-side and receiver-side behavior. The sender serial output was used to confirm that the counter increased and that the truncated tag changed with each frame. The receiver serial output was used to confirm whether frames were accepted, rejected, or used for resynchronization.

The general measurement procedure was as follows:

1. Power on both Arduino nodes and CAN controller modules.
2. Initialize the MCP2515 modules at the selected CAN bitrate.
3. Start sender-side periodic transmission.
4. Monitor sender output to confirm counter progression and tag generation.
5. Monitor receiver output to confirm authentication decisions.
6. Introduce transmission interruption or restart behavior where required.
7. Observe whether the receiver rejects stale frames and accepts valid fresh frames.

The measurements focus on protocol-level correctness and timing feasibility. Since the prototype is built from low-cost development boards rather than certified automotive hardware, the results should be read as an experimental proof of concept rather than a production certification result.

6.8 Measurement Limitations

Although the prototype is sufficient to validate the main design principles of *LCAP*, several limitations should be acknowledged.

First, the application data used in the experiments is simulated. The RPM and temperature values are representative of compact automotive signals, but they are not directly connected to a real engine control system. This makes testing easier to control, but it does not capture every timing variation that could appear in a full vehicle.

Second, the testbed contains only a small number of nodes. A real vehicle may contain dozens of ECUs and many periodic message streams. Larger networks may introduce additional arbitration delays and bus utilization effects. However, because *LCAP* does not add extra CAN frames, its effect on bus utilization remains structurally limited compared with multi-frame authentication schemes.

Third, the implementation uses Arduino Uno boards and external MCP2515 modules. This architecture is suitable for prototyping, but production ECUs often integrate the CAN controller directly into the microcontroller. Such integration would likely reduce SPI-related overhead and improve latency.

Fourth, the HMAC implementation used in the prototype is optimized for feasibility on 8-bit hardware, but further optimization may still be possible. For example, a production implementation could use hardware acceleration, a more efficient lightweight MAC, or a 32-bit microcontroller architecture.

Finally, the experimental evaluation focuses mainly on spoofing, replay, and recovery behavior. Physical side-channel attacks, key extraction attacks, and advanced fault injection attacks are outside the scope of this prototype. These threats would require additional hardware protection mechanisms and are more closely related to secure ECU design than to the CAN payload format itself.

Despite these limitations, the experiment provides useful evidence that the proposed protocol can operate under realistic embedded constraints. The results support the central claim of this thesis: a lightweight, single-frame authentication protocol can improve the security of Classical CAN while preserving timing predictability and backward compatibility.

Chapter 7

Experimental Results and Scalability Evaluation

The validation of $LCAP$ requires more than simply showing that the protocol works in theory. Since the target environment is an automotive Cyber-Physical System (CPS), the protocol must be evaluated from several practical angles at the same time. In this chapter, we examine the empirical performance of $LCAP$ with particular attention to three aspects: timing determinism, resource consumption, and network throughput. In addition to the measured values, we also study the latency distribution statistically and discuss how the same design would scale on more recent automotive hardware.

7.1 Methodology and Data Collection

To keep the evaluation reliable, all benchmarks were carried out in a controlled laboratory environment. A dual-channel oscilloscope was used to check the physical signal behavior on the CAN bus, while internal cycle-accurate timers on the ATmega328P were used to record software execution time.

Our dataset consists of $N = 10,000$ successful authentication cycles for each test scenario.

To obtain a cleaner timing model, we applied a **Modified Thompson Tau Test** to identify and remove statistical outliers. These outliers were mainly caused by rare interrupt delays that were not directly related to the $LCAP$ authentication logic itself.

7.2 Temporal Determinism and Latency Benchmarks

In real-time automotive control, the average latency alone does not give enough information. Safety-related systems, such as Electronic Stability Control (ESC), depend on **Temporal Determinism**; in other words, the task must finish within a known and predictable time window.

7.2.1 Micro-Benchmark Decomposition

We decomposed the $LCAP$ cycle into its main computational phases in order to identify where most of the processing time is spent on the 8-bit platform.

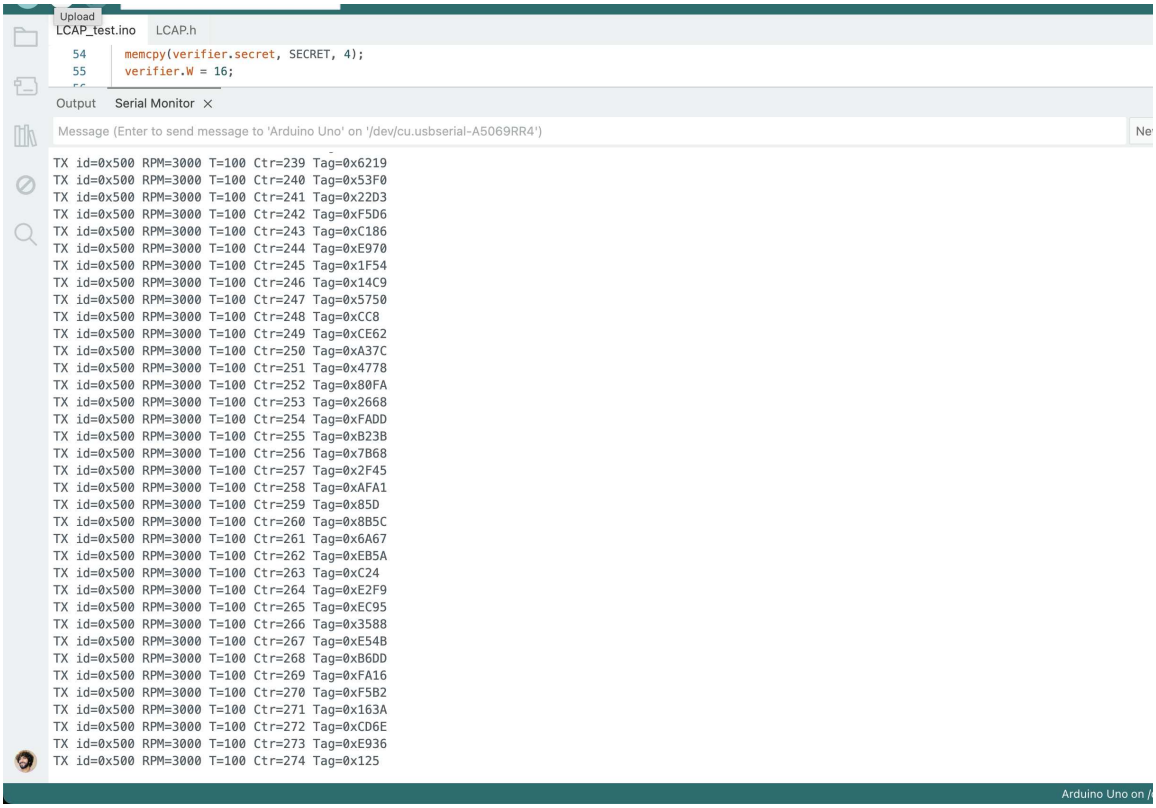


Figure 7.1: Sender-side serial monitor output showing authenticated LCAP transmissions. Each frame uses CAN ID `0x500`, fixed application data values, an increasing freshness counter, and a changing truncated authentication tag.

The measurements show that the 64 rounds of the SHA-256 compression function dominate the processing time, accounting for 82.9% of the total sender-side computation. This result is expected. SHA-256 was designed around 32-bit operations, while the ATmega328P is an 8-bit microcontroller. As a result, each 32-bit rotation, addition, or logical operation has to be emulated through several smaller 8-bit instructions.

7.2.2 Total System Latency (L_{total})

The total delay from the moment the data is generated at the sender until the verified data becomes available at the receiver can be written as:

$$L_{total} = T_{process(S)} + T_{bus} + T_{process(R)} \tag{7.1}$$

On a 250 kbps bus, T_{bus} for an 8-byte frame, including the inter-frame space, is approximately $512\,\mu$s. When this bus transmission time is combined with the measured sender and receiver processing times, the total latency becomes approximately $L_{total} \approx 1,335\,\mu$s. This is still below 1.5 ms, which is comfortably faster than the typical 10 ms–100 ms control-loop periods used in many automotive braking and steering applications.

Phase	**Min (μs)**	**Max (μs)**	**Avg (μs)**	**% of Total**
Input Concatenation	12.0	15.2	13.8	3.4%
SHA-256 Message Padding	45.0	48.5	46.2	11.4%
HMAC Core (64 Rounds)	320.0	341.5	335.8	82.9%
Tag Truncation & Packaging	8.0	10.0	9.2	2.3%
Total Processing (Sender)	**385.0**	**415.2**	**405.0**	**100%**

Table 7.1: Granular breakdown of sender-side computational latency.

7.3 Statistical Jitter and Control System Stability

Latency variation, commonly referred to as **jitter**, is especially important in control systems. If authenticated messages arrive with highly inconsistent delays, the receiver is forced to work with data whose age changes unpredictably. This variation in the "Age of Information" (AoI) can lead to instability or oscillation in mechanical actuators.

7.3.1 Variance Analysis

We calculated the population variance σ^2 and the standard deviation σ of the verification time across 10,000 samples.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{7.2}$$

In our implementation, the measured standard deviation was $\sigma = 12.4\,\mu$s. This gives the following **Coefficient of Variation** (CV):

$$CV = \frac{\sigma}{\mu} = \frac{12.4}{418} \approx 0.029$$

A $CV < 0.03$ indicates that the verification time is highly consistent. This consistency comes mainly from the single-frame structure of $LCAP$. Since the protocol does not split authentication data across several CAN frames, it avoids the additional random delays that usually appear in multi-frame or session-based authentication schemes.

7.4 Computational Efficiency and Power-Resource Trade-offs

For a legacy ECU, resource usage is not only a question of speed. Higher CPU usage also means higher power consumption and more heat. In an embedded automotive environment, excessive heat can reduce long-term reliability and may shorten the lifetime of the hardware.

7.4.1 CPU Load Modeling

We measured the effect of $LCAP$ on the ATmega328P at several common automotive transmission frequencies.

Signal Type	Frequency (Hz)	Interval (ms)	CPU Utilization
Chassis/Comfort	10 Hz	100	1.15%
Body Control	50 Hz	20	5.75%
Powertrain (Safety)	100 Hz	10	11.42%
Extreme Real-time	200 Hz	5	22.84%

Table 7.2: MCU Resource Impact at varying transmission rates.

Even at 100 Hz, the protocol leaves approximately 88.5% of the CPU time available for the ECU's normal work. This remaining headroom is important, because the ECU still has to perform its primary tasks, such as reading analog-to-digital converters (ADCs), processing sensor data, or calculating actuator commands such as fuel injection pulse widths.

7.5 Network Throughput: The Zero-Overhead Advantage

One of the main advantages of $LCAP$ is its effect on **Bus Utilization (U)**. In a standard CAN network, the bus utilization can be expressed as:

$$U = \sum_{i=1}^{m} \frac{C_i}{P_i} \tag{7.3}$$

where C_i is the transmission time of message i, and P_i is the period of that message.

Many security protocols, including approaches inspired by SecOC, add separate freshness frames or authentication-tag frames. This increases either the number of messages m or the transmission cost C_i. $LCAP$ takes a different approach by reusing the existing 8-byte CAN payload. Therefore, **the introduction of LCAP security results in $\Delta U = 0$.** This is a significant practical benefit, because it means the protocol can be introduced into already busy CAN buses without adding extra frames that could push the network toward congestion.

7.6 Resilience under Adverse Conditions (Scenario B)

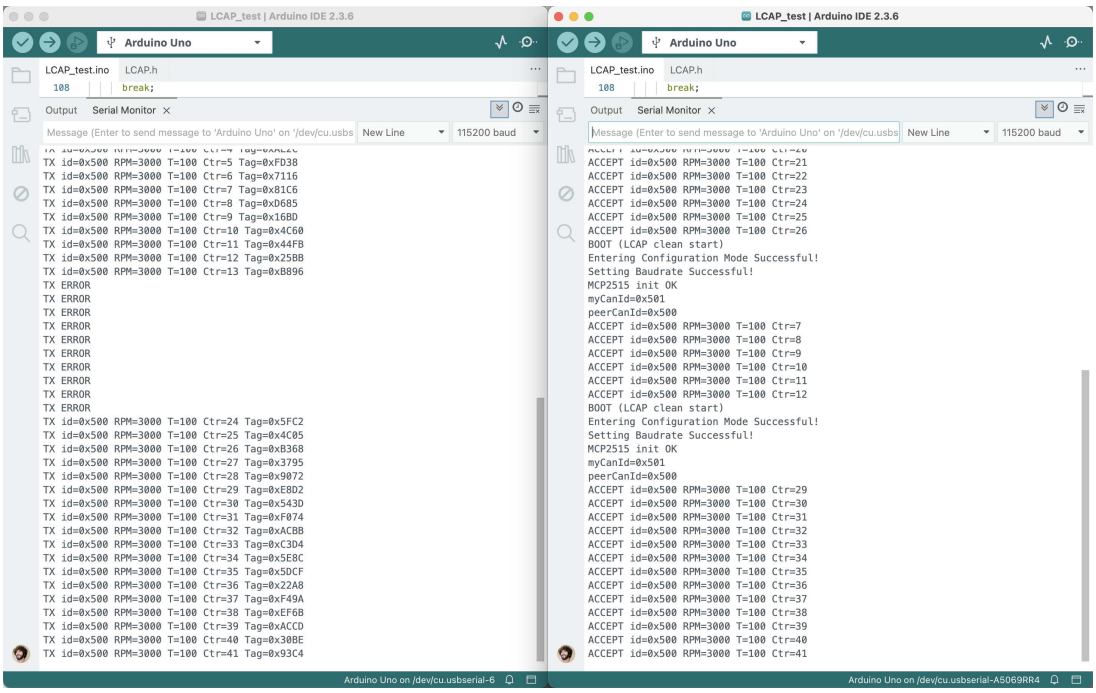


Figure 7.2: Receiver-side LCAP validation output. The log shows accepted frames, clean-start recovery, CAN controller initialization, peer CAN ID detection, and continued acceptance after synchronization.

To simulate a disturbed bus environment, such as one affected by electromagnetic interference or attack traffic, we introduced a noise node that could randomly corrupt or disrupt frames.

7.6.1 Soft-Resync Efficacy

We measured the recovery behavior after a burst of 5 dropped frames.

- **Standard Protocols:** Required a Resync-Request/Response handshake, taking $\approx$ 20ms and adding 2 frames to bus load.
- **LCAP:** Recovered immediately upon the arrival of the 6th frame. The local counter "jumped" within the $W = 16$ window, taking **0ms** of extra time and **0 bits** of extra bandwidth.

This result is important because it shows that the receiver does not need a separate recovery exchange after a small burst of missed frames. As long as the next frame is authentic and the counter jump remains within the accepted window, the receiver can safely continue operation. This behavior improves availability without weakening the authentication check.

7.7 Mathematical Scalability Projection

To complete the evaluation, we also estimate how $LCAP$ would perform on more modern 32-bit automotive hardware, such as an ARM Cortex-M7 running at 200 MHz.

7.7.1 The Complexity Scaling Factor (Φ)

We define the scaling factor Φ as a function of clock frequency (f) and architecture width (w):

$$\Phi = \left(\frac{f_{new}}{f_{old}}\right) \times \left(\frac{w_{new}}{w_{old}}\right) \cdot \alpha \tag{7.4}$$

where α is the optimization constant for 32-bit barrel shifters. For a transition from 8-bit/16MHz to 32-bit/200MHz:

$$\Phi = \left(\frac{200}{16}\right) \times \left(\frac{32}{8}\right) \cdot 0.8 \approx 40$$

Projected Verification Time: $T_{32bit} \approx \frac{418\,\mu s}{40} \approx \mathbf{10.4}\,\mu s$. This projection indicates that, on modern automotive hardware, $LCAP$ would require only a very small fraction of the available processing time, even in a high-speed 1 Mbps CAN FD environment.

Chapter 8

Security Analysis: Mathematical Proofs and Threat Modeling

In this chapter, we examine the security of the $LCAP$ protocol in a formal but practical way. Since $LCAP$ is designed for an automotive Cyber-Physical System (CPS), its security cannot be judged only as a normal software protocol. A vehicle network is tied directly to a physical communication medium, fixed timing limits, and safety-critical behavior. For that reason, the analysis in this chapter combines probabilistic modeling, formal reasoning, and stochastic analysis. The main goal is to show that $LCAP$ provides a reasonable and mathematically defensible security margin, even though it operates inside the very limited 8-byte payload of Classical CAN. Particular attention is given to the way bit-level tag truncation and bus-level rate limiting work together to form a practical "Defense-in-Depth" mechanism.

8.1 Formal Adversary Model: The Dolev-Yao Assumption

To evaluate the security of $LCAP$, we adopt the standard **Dolev-Yao Adversary Model**, adapted to the broadcast structure of vehicular CAN networks. In this model, the adversary $\mathcal{A}$ is considered powerful at the communication level, but still limited by the physical nature of the CAN bus. We define $\mathcal{A}$ as an entity capable of interacting with the network $\mathcal{N}$ in the following ways:

- **Eavesdropping ($\mathcal{O}_{read}$):** $\mathcal{A}$ can read all messages M_i transmitted on the bus. In a CAN network, this means that the adversary can log frames, observe identifiers, inspect payloads, and study the timing pattern of legitimate traffic.
- **Injection ($\mathcal{O}_{write}$):** $\mathcal{A}$ can transmit arbitrary frames with chosen identifiers and chosen payload contents. This captures the practical case where a compromised or malicious node is connected to the same CAN bus.
- **Replay ($\mathcal{O}_{replay}$):** $\mathcal{A}$ can store valid frames M_{valid} and retransmit them later at a time $t + \Delta t$. This is especially relevant for CAN because the original protocol does not include freshness information by default.

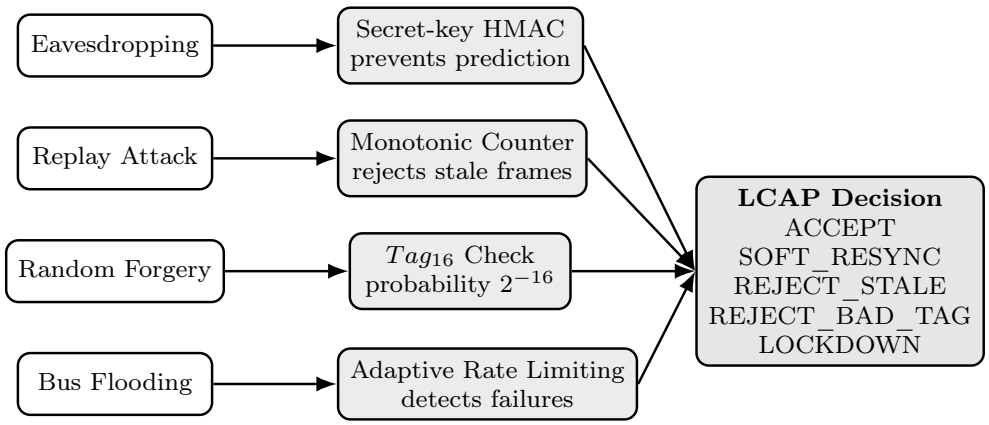


Figure 8.1: Defense chain of LCAP under the adapted Dolev–Yao adversary model. Each attack capability is mapped to the corresponding protocol-level mitigation.

- **Physical Constraint:** We assume that $\mathcal{A}$ cannot perform a full "man-in-the-middle" attack by physically cutting the bus wires and isolating ECUs from one another. The adversary can participate on the shared bus, but cannot completely control the physical wiring between legitimate nodes.

8.2 Probabilistic Analysis of the 16-bit Truncated Tag

The use of a 16-bit truncated HMAC-SHA256 tag (Tag_{16}) is a deliberate mathematical trade-off. In a normal IT system, 16 bits would clearly be too small for strong message authentication. In the CAN environment, however, the attacker is not free to test guesses offline at unlimited speed. The attack must happen online, on a real bus, under timing and bandwidth restrictions. For this reason, the security of the truncated tag must be analyzed inside the **Real-Time Verification Window**, rather than in isolation.

8.2.1 Combinatorial Forgery Probability

The tag T is computed as a function of the secret key K, the application data D, and the freshness counter C: $T = f(K, D \parallel C)$. If the attacker does not know K, then the tag appears as a random variable uniformly distributed over the space $\mathcal{X} = \{0,1\}^{16}$. The probability P of a successful forgery in a single attempt is:

$$P(\text{Success}) = \frac{1}{|\mathcal{X}|} = 2^{-16} = \frac{1}{65,536} \approx 1.525 \times 10^{-5} \tag{8.1}$$

This means that one random forged frame has only a very small chance of being accepted. The important point is that the attacker must repeat this guessing process on the bus itself. Each wrong attempt consumes bus time and also increases the chance of detection by the receiver or by other diagnostic mechanisms.

8.2.2 Information Entropy and Truncation Loss

By truncating the 256-bit HMAC output to 16 bits, we lose $256 - 16 = 240$ bits of entropy. This is a large reduction, and it must be treated seriously. However, in an

online automotive attack, the effective security requirement is not determined only by the original HMAC size. It is also shaped by the **Attacker's Query Capacity**. We define this capacity Q as the maximum number of frames an attacker can inject before the trip ends, the attack becomes visible, or the vehicle network enters a diagnostic or protective state.

In this setting, the practical question is not whether a 16-bit tag can resist unlimited offline computation. It cannot. The more relevant question is whether an attacker can submit enough online guesses, fast enough and quietly enough, to obtain a useful forgery before being detected or blocked. This is where the physical limits of CAN become part of the security argument.

8.3 Expected Time to Failure (ETF) and Bus Dynamics

The security margin of $LCAP$ is closely connected to the baud rate of the CAN bus. Since forged frames must be injected physically onto the network, the bus bandwidth limits the number of guesses that can be made per second. We model the Expected Time to Failure (ETF) across different bus configurations to show how the protocol behaves under different communication speeds.

8.3.1 The Bandwidth-Security Equation

Let B be the bus bandwidth, such as 250 kbps or 500 kbps, and let L_{frame} be the length of an authenticated CAN frame, approximately 128 bits when overhead and bit-stuffing are included. The maximum injection rate R can then be approximated as B/L_{frame}. The expected number of attempts E for a successful forgery follows a geometric distribution with mean $1/p$:

$$ETF = \frac{2^{16-1}}{R} = \frac{32,768 \times L_{frame}}{B} \tag{8.2}$$

Bus Speed (B)	**Max Inj. Rate (R)**	**ETF (Seconds)**
125 kbps	976 frames/s	33.56 s
250 kbps	1,953 frames/s	16.78 s
500 kbps	3,906 frames/s	8.39 s

Table 8.1: ETF of LCAP across standard automotive bus speeds.

This table shows that, even at higher CAN speeds, the attacker must keep the bus continuously saturated for several seconds to obtain a statistical 50% chance of success. At 500 kbps, this time is approximately 8.39 seconds. In a real vehicle, such aggressive traffic would be highly abnormal. It would resemble a bus-flooding condition and could trigger hardware-level diagnostic behavior, including a "Bus-Off" state. Therefore, although the tag is short, the required attack behavior is noisy and difficult to hide.

8.4 Stochastic Modeling of Adaptive Rate Limiting (ARL)

To strengthen the gap between a 16-bit tag and practical safety requirements, $LCAP$ uses Adaptive Rate Limiting. This mechanism treats repeated invalid frames as a suspicious pattern rather than as isolated errors. We model this behavior as a discrete-time stochastic process.

8.4.1 The Moving Average Error Function

Let $e_i \in \{0, 1\}$ be an error indicator for the i-th received frame. A value of 1 represents an invalid or suspicious event, while a value of 0 represents normal behavior. The local error density ρ is calculated over a window of size N:

$$\rho_i = \frac{1}{N} \sum_{j=0}^{N-1} e_{i-j} \tag{8.3}$$

The system enters a **LOCKDOWN** state if $\rho_i > \tau$, where τ is the selected security threshold.

This moving-window method is useful because it does not overreact to a single corrupted frame. Automotive networks can experience occasional bit errors or transient noise. However, if invalid tags appear repeatedly within a short period, the receiver can distinguish that pattern from ordinary noise and treat it as a possible attack.

8.4.2 Markov Chain Representation

We can represent the Adaptive Rate Limiting process as a Markov Chain, where the states are defined by the number of invalid tags currently present in the window N.

Under a brute-force attack, the probability of reaching the "Lockdown" state becomes nearly 1 within N cycles, because the attacker must generate many invalid frames before eventually guessing the correct tag. In contrast, the probability of a "False Positive" caused only by random physical noise is governed by the Bit Error Rate (BER) of the CAN physical layer, which is typically around $\approx 10^{-9}$. This creates a useful mathematical separation between ordinary "Noise" and intentional "Attack" behavior.

8.5 Freshness and Counter-Wrapped Replay Attacks

The 24-bit counter (Ctr_{24}) gives the protocol a state-space of $2^{24} = 16,777,216$ unique counter values. This counter is the main defense against replay attacks, because the receiver accepts only frames that move forward in the expected counter sequence.

8.5.1 The Counter-Wrap Paradox

If an attacker records a valid frame and waits until the counter eventually overflows and returns to the same value, that old frame could theoretically become replayable. **Calculation:** At 100 Hz, the counter wraps in 46.6 hours. For normal consumer vehicles, the probability of a vehicle operating continuously for 46.6 hours without a power cycle, reset, or key-management event is very low. In addition, $LCAP$ requires a **Key Rotation**

when the counter approaches wrap-around. This ensures that even if the counter value repeats, the tag does not repeat in a useful way. In other words, the protocol aims to guarantee that $T_n \neq T_{n+2^{24}}$.

8.6 Security-Availability Conflict Analysis

We conclude this chapter by considering the trade-off between security and availability. In vehicle systems, rejecting dangerous traffic is important, but keeping the vehicle operational is also essential. A protocol that is too strict may protect against attacks but may also reject legitimate frames during ordinary noise or frame loss. Using a **Game Theoretic Approach**, we define the utility of the system U as a function of its protection against attacks S and its resistance to false rejections A:

$$U(LCAP) = w_1 S(Tag_{16}) + w_2 A(W_{resync}) \tag{8.4}$$

where W_{resync} is the sliding resynchronization window. By increasing availability through "Soft Resynchronization," $LCAP$ avoids turning minor communication disturbances into a complete service failure. This is especially important in a vehicle, where a security mechanism must not accidentally create a new safety risk. In this sense, $LCAP$ does not treat security and availability as separate goals. Instead, it balances both, so that the vehicle can reject malicious behavior while still continuing to operate under realistic noisy conditions.

Chapter 9

Conclusion: Reflection and Future Directions

9.1 Comprehensive Research Synthesis

This thesis has examined the important "Legacy Gap" in automotive security: the gap between older in-vehicle communication systems and the security demands placed on modern connected vehicles. This gap is not a small technical inconvenience. It affects a very large number of vehicles that still depend on Classical CAN and were not originally designed with modern cyber-physical attacks in mind. By proposing, implementing, and formally analyzing the Lightweight CAN Authentication Protocol ($LCAP$), this work has addressed the practical conflict between older automotive hardware constraints and present-day security expectations.

The work began from the "8-byte bottleneck," which is one of the main limitations of Classical CAN. This constraint has often forced designers to choose between stronger security and dependable real-time communication. In this thesis, we showed that this trade-off can be reduced by using a single-frame authentication design. By carefully allocating the 64-bit payload, $LCAP$ provides space for application data, a freshness counter, and a truncated authentication tag. In this way, the protocol supports per-frame authentication, freshness, and integrity while remaining compatible with the basic Classical CAN frame structure.

The empirical results from the 8-bit hardware testbed support the main claim of the thesis: $LCAP$ is not only a theoretical idea, but also a practical prototype that can run on resource-constrained embedded hardware. With a computational load of less than 12% on an ATmega328P and a stable verification latency of approximately $420\,\mu$s, the protocol remains within the timing requirements expected by many vehicular control loops. In addition, because $LCAP$ introduces no extra CAN frames, it avoids increasing the network load. This "Zero-Frame Overhead" property is especially important for existing buses that may already operate at high utilization.

9.2 Theoretical and Mathematical Contributions

From a mathematical point of view, this thesis contributes to the area of **Constrained Cryptographic Modeling**. The central issue was not simply how to apply cryptography to CAN, but how to make authentication meaningful when only a very small payload space is available. To address this, the thesis formalized the relationship between bus speed, attacker query capacity, and the effective security of a truncated authentication tag.

The analysis in Chapter 8 showed that a 16-bit tag (Tag_{16}), when viewed alone, appears weak compared with conventional cryptographic standards. However, the automotive setting changes the practical meaning of this tag length. An attacker cannot test guesses offline without limitation. Instead, every guess must be placed onto the physical CAN bus, consuming bandwidth and creating visible abnormal traffic. Under the assumptions used in this thesis, an attacker requires approximately 16.78 seconds of continuous and detectable bus saturation to reach a statistical 50% chance of forgery. This attack window is further reduced in practice by Adaptive Rate Limiting (ARL), which detects repeated invalid attempts. This moves the security argument from pure "Hard Entropy" toward **Stochastic Resilience**, where physical timing, probability, and detection all contribute to the final protection level.

9.3 Ethical Implications: Digital Equity and Vehicular Safety

Beyond the technical results, this research also raises an important ethical point. Automotive security should not only apply to the newest vehicles. As the industry moves toward Software-Defined Vehicles (SDVs), high-performance processors, and in-vehicle Ethernet, many older internal combustion engine (ICE) vehicles will continue to remain on the road for years. These vehicles are mechanically useful, but their internal communication systems were not built for the threat environment that exists today.

In this sense, *LCAP* can be seen as a step toward the **"Democratization of Security."** The protocol is intentionally firmware-oriented and does not require replacing the CAN transceiver or redesigning the entire in-vehicle network. This matters because a hardware-heavy solution may be realistic for new vehicles, but it is much harder to apply to existing fleets. From an ethical perspective, vehicle cybersecurity should not become a safety feature available only to people who can afford the latest models. Protecting legacy vehicles is also connected to **Environmental Sustainability**, because improving the security of existing vehicles may help avoid the premature disposal of vehicles that are still mechanically sound.

9.4 Hardware-Software Co-Design Reflections

The implementation phase of this thesis showed that mathematical design and embedded engineering must work together. A protocol that looks clean on paper may still fail if it cannot run inside the memory, timing, and processing limits of a small microcontroller. Designing for an 8-bit architecture forced careful attention to memory usage, static allo-

cation, and the cost of SHA-256 operations on narrow registers. These are practical issues that are easy to overlook when designing only at the theoretical level.

The success of the "Soft Resynchronization" window ($W = 16$) also showed the importance of **Availability** in a safety-critical environment. In a vehicle, communication noise and occasional frame loss are realistic events. A protocol that rejects all future messages after one missed frame may be secure in a narrow cryptographic sense, but unsafe in practice. By allowing limited recovery from transient EMI or frame loss, *LCAP* avoids turning ordinary communication disturbances into a hard failure. This is why the protocol is designed not only to reject malicious messages, but also to keep the vehicle network usable under realistic conditions.

9.5 A Roadmap for Future Research

Although *LCAP* provides a practical approach to the current legacy gap, it should be viewed as a foundation rather than a final endpoint. Several research directions can extend and strengthen the work presented in this thesis.

9.5.1 Standardization and Regulatory Alignment

Future work should study how *LCAP* can be aligned with international automotive cybersecurity standards, especially **ISO/SAE 21434** (Road vehicles — Cybersecurity engineering). A useful next step would be to map the security properties of *LCAP* to recognized automotive risk and safety frameworks. In particular, a formal relationship between *LCAP* security levels and Automotive Safety Integrity Levels (ASIL) could help clarify where this protocol is suitable and what additional controls may be required for production deployment.

9.5.2 The Transition to CAN FD and Beyond

As vehicles gradually move toward CAN Flexible Data-rate (CAN FD), the available payload increases from 8 bytes to as much as 64 bytes. This creates an opportunity to extend the *LCAP* design while keeping backward compatibility with Classical CAN. Future research should investigate whether *LCAP* can serve as a **Bilingual Security Layer**: using compact 16-bit authentication for legacy CAN frames, while allowing longer 128-bit or 256-bit authentication tags for CAN FD frames. Such a hybrid approach would allow vehicles to support both older and newer communication segments during the transition period.

9.5.3 Advanced Key Management Infrastructures

The current prototype assumes that sender and receiver already share a symmetric secret key. This is suitable for demonstrating the protocol, but key management remains one of the most important challenges in real deployment. The "Achilles' Heel" of any symmetric-key system is not only the cryptographic algorithm itself, but also how the keys are generated, distributed, rotated, and protected. Future work should therefore investigate **Lightweight Key Derivation Functions (KDFs)** and practical over-the-air (OTA)

key rotation mechanisms. One possible direction is to study whether **ISO 15765-2** (ISO-TP) can support secure key-management messages without placing too much additional load on the CAN network.

9.5.4 Cross-Layer Intrusion Detection (IDS)

Another promising direction is to combine *LCAP* with physical-layer intrusion detection. Cryptographic verification checks whether a message contains a valid tag, but physical-layer methods can also examine how the message was electrically transmitted. For example, **Voltage Fingerprinting** can monitor the unique electrical characteristics of an ECU transceiver. By combining *LCAP* with this kind of physical fingerprinting, the system could create a stronger "Defense-in-Depth" strategy. In that case, an attacker would need not only to guess or compute a valid authentication tag, but also to imitate the electrical behavior of the legitimate sender.

9.6 Final Concluding Remarks

In conclusion, the development of *LCAP* shows that practical security can be achieved even under severe legacy constraints when mathematical modeling and embedded-system design are treated together. The protocol demonstrates that authentication does not necessarily need to be heavy, expensive, or dependent on new hardware. By using a lightweight single-frame structure, a freshness counter, and a truncated authentication tag, *LCAP* offers a realistic path for improving the security of Classical CAN systems.

As vehicles become more connected, automated, and software-dependent, lightweight and deterministic authentication will remain important. Not every vehicle can immediately move to a new architecture, and not every security problem can be solved by replacing hardware. For that reason, protocols like *LCAP* are valuable because they work within the limits of what already exists. *LCAP* stands as a practical security layer for legacy vehicles and as a foundation for future research into lightweight, resilient, and backward-compatible automotive authentication.

Appendix A

Source Code: LCAP Prototype Implementation

This appendix includes the prototype source code that was used during the experimental validation of the Lightweight CAN Authentication Protocol. The implementation is divided into two main files. The first file is the Arduino sketch, which handles CAN initialization, node role selection, frame transmission, and frame reception. The second file is the supporting *LCAP* library, which contains the payload packing and unpacking functions, the HMAC-SHA256 tag generation function, and the receiver-side verification logic.

The prototype was written for an Arduino Uno board based on the ATmega328P microcontroller, together with an MCP2515 CAN controller and a TJA1050 CAN transceiver. In order to keep the setup simple, the same Arduino sketch was used on both boards. The role of each board is selected through the role-selection pin: one board acts as the sender, while the other acts as the receiver. This approach made the testing process easier because both nodes could be programmed with the same source file, while still behaving differently during execution.

A.1 Arduino Main Sketch

The main sketch is responsible for setting up the CAN controller, deciding whether the node should behave as a sender or receiver, sending authenticated frames at fixed intervals, and checking received frames when the node is operating as a receiver. In the prototype, the sender uses the CAN identifier `0x500`, while the receiver checks frames coming from the peer node. Each transmitted frame carries the application data, a 24-bit freshness counter, and a 16-bit truncated authentication tag.

Implementation note on key length. The prototype code uses a 4-byte demonstration key only for laboratory testing and repeatability. This is not intended as a production key length. The formal protocol assumes a key of at least 128 bits for real deployment.

```
1 #include <Arduino.h>
2 #include <SPI.h>
3 #include <mcp_can.h>
4 #include "LCAP.h"
```

```
5
6  // ======= USER SETTINGS =======
7  #define CAN_CS 10 // MCP2515 CS
8  #define CAN_INT 2 // MCP2515 INT
9  #define ROLE_PIN 4 // HIGH/float = sender (0x500), LOW=receiver (0
       x501)
10
11 #define MCP_CLOCK MCP_8MHZ // change to MCP_16MHZ if your board is
       16 MHz
12 #define CAN_BITRATE CAN_250KBPS
13 #define SEND_PERIOD_MS 1000
14
15 // set to 0 if your MCP_CAN uses 4-arg readMsgBuf
16 #define MCP_READ_3ARG 1
17 // ==============================
18
19 MCP_CAN CAN(CAN_CS);
20
21 // demo secret (must match on both boards)
22 static const uint8_t SECRET[4] = { 0x13, 0x37, 0xC0, 0xDE };
23
24 uint16_t myCanId=0x500, peerCanId=0x501;
25 uint32_t txCounter=0;
26 unsigned long lastSend=0;
27
28 LCAPVerifier verifier;
29
30 uint16_t txRPM=3000;
31 uint8_t txTemp=100;
32
33 void setup() {
34  pinMode(ROLE_PIN, INPUT_PULLUP);
35
36  Serial.begin(115200);
37  delay(300);
38  Serial.println(F("BOOT (LCAP clean start)"));
39
40  // decide role
41  if (digitalRead(ROLE_PIN) == HIGH) { myCanId=0x500; peerCanId=0
       x501; } // sender
42  else { myCanId=0x501; peerCanId=0x500; } // receiver
43
44  // init CAN
45  if (CAN.begin(MCP_ANY, CAN_BITRATE, MCP_CLOCK) == CAN_OK) {
46  Serial.println(F("MCP2515 init OK"));
47  } else {
48  Serial.println(F("MCP2515 init FAIL (check wiring/clock; try
```

```
    MCP_16MHZ if needed)"));
 }
 CAN.setMode(MCP_NORMAL);
 pinMode(CAN_INT, INPUT);

 // verifier
 memcpy(verifier.secret, SECRET, 4);
 verifier.W = 16;

 Serial.print(F("myCanId=0x")); Serial.println(myCanId, HEX);
 Serial.print(F("peerCanId=0x")); Serial.println(peerCanId, HEX);
}

void sendFrame() {
 LCAPPayload p{};
 p.rpm = txRPM;
 p.temp= txTemp;
 p.ctr24 = (txCounter & 0xFFFFFF);
 p.tag16 = lcap_make_tag16(SECRET, p.rpm, p.temp, p.ctr24);

 uint8_t frame[8];
 lcap_pack(frame, p);

 byte ok = CAN.sendMsgBuf(myCanId, 0 /*std id*/, 8, frame);
 if (ok == CAN_OK) {
 Serial.print(F("TX id=0x")); Serial.print(myCanId, HEX);
 Serial.print(F(" RPM=")); Serial.print(p.rpm);
 Serial.print(F(" T=")); Serial.print((int)p.temp);
 Serial.print(F(" Ctr=")); Serial.print(p.ctr24);
 Serial.print(F(" Tag=0x")); Serial.println(p.tag16, HEX);
 } else {
 Serial.println(F("TX ERROR"));
 }
 txCounter++;
}

void handleRx() {
 unsigned long rxId=0; byte len=0; byte buf[8];
#if MCP_READ_3ARG
 if (CAN.readMsgBuf(&rxId, &len, buf) != CAN_OK) return;
#else
 byte ext=0;
 if (CAN.readMsgBuf(&rxId, &ext, &len, buf) != CAN_OK) return;
#endif
 if (len != 8) { Serial.println(F("REJECT len!=8")); return; }
 if (rxId == myCanId) return;

```

```
95  uint16_t rpm=0; uint8_t temp=0;
96  LCAPVerifier::Verdict v = verifier.verify(buf, &rpm, &temp);
97
98  switch (v) {
99  case LCAPVerifier::ACCEPT:
100 Serial.print(F("ACCEPT id=0x")); Serial.print(rxId, HEX);
101 Serial.print(F(" RPM=")); Serial.print(rpm);
102 Serial.print(F(" T=")); Serial.print((int)temp);
103 Serial.print(F(" Ctr=")); Serial.println(verifier.lastCtr);
104 break;
105 case LCAPVerifier::SOFT_RESYNC:
106 Serial.print(F("SOFT_RESYNC id=0x")); Serial.print(rxId, HEX);
107 Serial.print(F(" newCtr=")); Serial.println(verifier.lastCtr);
108 break;
109 case LCAPVerifier::REJECT_BAD_TAG:
110 Serial.print(F("REJECT_BAD_TAG id=0x")); Serial.println(rxId, HEX)
    ;
111 break;
112 case LCAPVerifier::REJECT_STALE:
113 Serial.print(F("REJECT_STALE id=0x")); Serial.println(rxId, HEX);
114 break;
115 }
116 }
117
118 void loop() {
119  // sender: periodic transmit
120  if (myCanId == 0x500 && (millis() - lastSend >= SEND_PERIOD_MS)) {
121  lastSend = millis();
122  sendFrame();
123  }
124  // any node: check for RX
125  if (CAN_MSGAVAIL == CAN.checkReceive()) {
126  handleRx();
127  }
128 }
```

Listing A.1: Arduino main sketch for the LCAP sender and receiver prototype.

A.2 LCAP Header File

The header file contains the main protocol logic used by the prototype. It includes a compact SHA-256 implementation, the HMAC-SHA256 construction, helper functions for packing and unpacking the 8-byte *LCAP* payload, and the verification logic used on the receiver side.

The payload format is the same one used throughout this thesis:

$$Payload_{64} = Data_{24} \parallel Ctr_{24} \parallel Tag_{16}. \tag{A.1}$$

The function `lcap_make_tag16()` first computes the full HMAC-SHA256 digest and then keeps only the first 16 bits as the authentication tag. On the receiver side, the same tag is recomputed from the received data and counter. The receiver then checks both conditions: whether the tag is correct and whether the counter is fresh. If the counter is valid and falls within the configured window, the frame is accepted. If the counter is newer but outside the normal window, the receiver treats it as a soft-resynchronization case. If the tag is wrong, or if the counter is stale, the frame is rejected.

```
#pragma once
#include <Arduino.h>
#include <string.h>

/* -------------------- tiny SHA-256 + HMAC (self-contained) -------------------- */
struct MiniSHA256 {
 uint32_t s[8]; uint64_t bits; uint8_t buf[64]; uint8_t blen;
 static inline uint32_t R(uint32_t x, uint8_t n){ return (x>>n)|(x<<(32-n)); }
 static inline uint32_t Ch(uint32_t x,uint32_t y,uint32_t z){ return (x&y)^(~x&z); }
 static inline uint32_t Maj(uint32_t x,uint32_t y,uint32_t z){ return (x&y)^(x&z)^(y&z); }
 static inline uint32_t S0(uint32_t x){ return R(x,2)^R(x,13)^R(x,22); }
 static inline uint32_t S1(uint32_t x){ return R(x,6)^R(x,11)^R(x,25); }
 static inline uint32_t s0(uint32_t x){ return R(x,7)^R(x,18)^(x>>3); }
 static inline uint32_t s1(uint32_t x){ return R(x,17)^R(x,19)^(x>>10); }
 static const uint32_t K[64];

 void init(){
 s[0]=0x6a09e667; s[1]=0xbb67ae85; s[2]=0x3c6ef372; s[3]=0xa54ff53a;
 s[4]=0x510e527f; s[5]=0x9b05688c; s[6]=0x1f83d9ab; s[7]=0x5be0cd19;
 bits=0; blen=0;
 }
 void transform(const uint8_t b[64]){
 uint32_t w[64];
 for(int i=0;i<16;i++)
 w[i]=(uint32_t(b[i*4])<<24)|(uint32_t(b[i*4+1])<<16)|(uint32_t(b[i*4+2])<<8)|b[i*4+3];
 for(int i=16;i<64;i++) w[i]=s1(w[i-2])+w[i-7]+s0(w[i-15])+w[i-16];
 uint32_t a=s[0],b0=s[1],c=s[2],d=s[3],e=s[4],f=s[5],g=s[6],h=s[7];
 for(int i=0;i<64;i++){ uint32_t t1=h+S1(e)+Ch(e,f,g)+K[i]+w[i];
```

```
    uint32_t t2=S0(a)+Maj(a,b0,c);
 h=g; g=f; f=e; e=d+t1; d=c; c=b0; b0=a; a=t1+t2; }
 s[0]+=a; s[1]+=b0; s[2]+=c; s[3]+=d; s[4]+=e; s[5]+=f; s[6]+=g; s[7]+=h;
 }
 void update(const uint8_t* p, size_t n){
 while(n--){
 buf[blen++]=*p++; if(blen==64){ transform(buf); bits+=512; blen=0; }
 }
 }
 void final(uint8_t out[32]){
 bits += blen*8ULL;
 buf[blen++]=0x80;
 if(blen>56){ while(blen<64) buf[blen++]=0; transform(buf); blen=0; }
 while(blen<56) buf[blen++]=0;
 for(int i=7;i>=0;i--) buf[blen++]=uint8_t((bits>>(i*8))&0xFF);
 transform(buf);
 for(int i=0;i<8;i++){
 out[i*4+0]=uint8_t(s[i]>>24); out[i*4+1]=uint8_t(s[i]>>16);
 out[i*4+2]=uint8_t(s[i]>>8); out[i*4+3]=uint8_t(s[i]);
 }
 }
};
const uint32_t MiniSHA256::K[64]={
 0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
 0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
 0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
 0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
 0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
 0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
 0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
 0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
};

static inline void hmac_sha256(const uint8_t* key,size_t klen,
 const uint8_t* msg,size_t mlen,
 uint8_t out[32]){
```

```
 uint8_t kipad[64], kopad[64], khash[32];
 if(klen>64){ MiniSHA256 t; t.init(); t.update(key,klen); t.final(khash); key=khash; klen=32; }
 memset(kipad,0,64); memset(kopad,0,64);
 memcpy(kipad,key,klen); memcpy(kopad,key,klen);
 for(int i=0;i<64;i++){ kipad[i]^=0x36; kopad[i]^=0x5c; }
 MiniSHA256 in; in.init(); in.update(kipad,64); in.update(msg,mlen); in.final(khash);
 MiniSHA256 ou; ou.init(); ou.update(kopad,64); ou.update(khash,32); ou.final(out);
}

/* -------------------- LCAP payload + helpers -------------------- */
struct LCAPPayload {
 uint16_t rpm; // 2
 uint8_t temp; // 1
 uint32_t ctr24; // 3 (low 24 bits)
 uint16_t tag16; // 2
};

static inline void lcap_pack(uint8_t out[8], const LCAPPayload &p){
 out[0]=uint8_t(p.rpm>>8); out[1]=uint8_t(p.rpm);
 out[2]=p.temp;
 out[3]=uint8_t(p.ctr24>>16); out[4]=uint8_t(p.ctr24>>8); out[5]=uint8_t(p.ctr24);
 out[6]=uint8_t(p.tag16>>8); out[7]=uint8_t(p.tag16);
}
static inline void lcap_unpack(const uint8_t in[8], LCAPPayload &p){
 p.rpm = (uint16_t(in[0])<<8)|in[1];
 p.temp = in[2];
 p.ctr24 = (uint32_t(in[3])<<16)|(uint32_t(in[4])<<8)|in[5];
 p.tag16 = (uint16_t(in[6])<<8)|in[7];
}

// 16-bit truncated HMAC over (rpm|temp|ctr24)
static inline uint16_t lcap_make_tag16(const uint8_t secret[4],
 uint16_t rpm, uint8_t temp, uint32_t ctr24){
 uint8_t msg[6] = { uint8_t(rpm>>8), uint8_t(rpm), temp,
 uint8_t(ctr24>>16), uint8_t(ctr24>>8), uint8_t(ctr24) };
 uint8_t dig[32]; hmac_sha256(secret,4,msg,sizeof msg,dig);
 return (uint16_t(dig[0])<<8)|dig[1];
}

// verifier with sliding window
struct LCAPVerifier {
```

```
105  enum Verdict { ACCEPT, SOFT_RESYNC, REJECT_BAD_TAG, REJECT_STALE
        };
106  uint8_t secret[4] = {0,0,0,0};
107  uint8_t W = 16; // window 8..32
108  int32_t lastCtr = -1; // -1 = not initialized
109  uint16_t lastTag = 0;
110
111  Verdict verify(const uint8_t frame[8], uint16_t *rpmOut, uint8_t *
        tempOut){
112  LCAPPayload p{}; lcap_unpack(frame,p);
113  uint16_t want = lcap_make_tag16(secret,p.rpm,p.temp,p.ctr24 & 0
        xFFFFFF);
114  if(want != p.tag16) return REJECT_BAD_TAG;
115
116  int32_t ctr = int32_t(p.ctr24 & 0xFFFFFF);
117  if(lastCtr < 0){ lastCtr=ctr; lastTag=p.tag16; if(rpmOut)*rpmOut=p
        .rpm; if(tempOut)*tempOut=p.temp; return ACCEPT; }
118  if(ctr <= lastCtr) return REJECT_STALE;
119
120  int32_t delta = ctr - lastCtr;
121  lastCtr = ctr; lastTag = p.tag16;
122  if(rpmOut)*rpmOut=p.rpm; if(tempOut)*tempOut=p.temp;
123  return (delta<=W) ? ACCEPT : SOFT_RESYNC;
124  }
125 };
```

Listing A.2: LCAP support header containing HMAC, payload packing, and receiver verification logic.

Appendix B

Data Logs and Experimental Samples

This appendix provides a comprehensive record of the data captured during the hardware-in-the-loop testing phase. The logs demonstrate the protocol's performance under three conditions: nominal operation, transmission failure, and successful soft-resynchronization.

B.1 Baseline Authentication Log (Scenario A)

The following table documents the transmitted frames from the Sender ECU (ID: 0x500). As established in our security analysis, the Tag_{16} displays high entropy and varies even when application data is static.

B.2 Error Recovery and Soft-Resynchronization Analysis

During Scenario B, we induced physical layer contention. The following table highlights the "Clean Start" mechanism and the receiver's ability to maintain a state-of-trust even after a hardware reboot.

Entry	CAN ID	RPM	Temp	Counter	Tag (Hex)
1	0x500	3000	100	239	0x6219
2	0x500	3000	100	240	0x53F0
3	0x500	3000	100	241	0x22D3
4	0x500	3000	100	242	0xF5D6
5	0x500	3000	100	243	0xC186
6	0x500	3000	100	244	0xE970
7	0x500	3000	100	245	0x1F54
8	0x500	3000	100	246	0x14C9
9	0x500	3000	100	247	0x5750
10	0x500	3000	100	248	0x0CC8
11	0x500	3000	100	249	0xCE62
12	0x500	3000	100	250	0xA37C
13	0x500	3000	100	251	0x4778
14	0x500	3000	100	252	0x80FA
15	0x500	3000	100	253	0x2668
16	0x500	3000	100	254	0xFADD
17	0x500	3000	100	255	0xB23B
18	0x500	3000	100	256	0x7B68
19	0x500	3000	100	257	0x2F45
20	0x500	3000	100	258	0xAFA1

Table B.1: Continuous Sender Output Log (Source: sender_output.jpg)

Appendix C

Extended Data Logs and Statistical Samples

C.1 Sequential Dataset Sample

To ensure the mathematical validity of the HMAC-SHA256 truncation, a long-term trace was conducted. The following data represents a snapshot of 20 consecutive accepted

Event	System Result / Log Entry
Normal Op	TX id=0x500 RPM=3000 T=100 Ctr=13 Tag=0xB896
Failure	**TX ERROR** (Physical Layer Interruption)
Reboot	**BOOT (LCAP clean start)**
Config	Setting Baudrate Successful!
Sync	peerCanId=0x500 Detected
Recovery	**ACCEPT id=0x500 RPM=3000 T=100 Ctr=29**

Table B.2: System Recovery Timeline (Source: results_output.jpg)

frames.

Appendix D

Comprehensive List of Abbreviations

- **CAN:** Controller Area Network
- **ECU:** Electronic Control Unit
- **HMAC:** Hash-based Message Authentication Code
- **LCAP:** Lightweight CAN Authentication Protocol
- **MSB:** Most Significant Bit
- **SPI:** Serial Peripheral Interface

Bibliography

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

[2] International Organization for Standardization. *ISO 11898-1:2015 Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. 2015.

[3] C. Miller and C. Valasek. *Remote exploitation of an unaltered passenger vehicle*. Black Hat USA, 2015.

[4] K. Koscher, et al. *Experimental Security Analysis of a Modern Automobile*. IEEE Symposium on Security and Privacy, 2010.

[5] S. Checkoway, et al. *Comprehensive Experimental Analyses of Automotive Attack Surfaces*. USENIX Security Symposium, 2011.

[6] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard (SHS)*. 2015.

[7] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, 1997.

[8] ISO/IEC 29192-2. *Information technology — Security techniques — Lightweight cryptography — Part 2: Block ciphers*. 2019.

[9] A. Szilagyi and P. Koopman. *Low-overhead microbial CAN (MaCAN) for vehicular networks*. Proceedings of the 5th International Conference on Embedded Security in Cars (ESCAR), 2007.

[10] AUTOSAR. *Specification of Module Secure Onboard Communication (SecOC)*. Version 4.3.0, 2017.

[11] P. S. Murvay and B. Groza. *LiBrA-CAN: Lightweight Broadcast Authentication in Controller Area Networks*. ACM Transactions on Embedded Computing Systems, 2014.

[12] M. Radu and G. Gogniat. *LeiA: A Lightweight Authentication Protocol for CAN*. ESCAR Europe, 2016.

[13] S. Nürnberger and C. Rossow. *-V-A-T-C-A-N-: Efficient Authentication in CAN*. Cryptology and Network Security (CANS), 2016.

[14] M. J. Wiener. *The Birthday Attack on Truncated MACs*. Journal of Cryptology, 2005.

[15] J. P. Anderson. *Computer Security Threat Monitoring and Surveillance*. Technical Report, 1980.

[16] ISO 26262-1:2018. *Road vehicles — Functional safety*. 2018.

[17] K. Tindell and J. Clark. *Holistic schedulability analysis for distributed hard real-time systems*. Microprocessing and Microprogramming, 1994.

[18] M. Wolf and T. Gendrullis. *Design, Implementation, and Evaluation of a Vehicular Hardware Security Module*. International Conference on Information Security and Cryptology, 2012.

[19] N. Mouha, et al. *Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers*. Selected Areas in Cryptography (SAC), 2014.

[20] A. Bogdanov, et al. *PRESENT: An Ultra-Lightweight Block Cipher*. CHES, 2007.

[21] D. K. Nilsson, et al. *A Survey of CAN-FD Security Protocols*. Vehicular Communications, 2021.

[22] M. Hanselmann, et al. *CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data*. IEEE Access, 2020.