Replay Attack Prevention in Smart Car IoT Systems:
A Lightweight Hybrid Nonce-Counter Framework

A report submitted in partial fulfilment of the requirements
for the Degree of Master in Information Technology
at Whitecliffe

By

KIRONRAJ ODATT PERINGODE
(Student ID: 20241136)

Supervisor: Dr. Muhammad Azam

IT9115 - Project

School of Information Technology
Whitecliffe, New Zealand

September 10, 2026

Dedication and Acknowledgements

To my family, whose endless support and sacrifice made this work possible — I dedicate this study.

I would like to express my sincere gratitude to my supervisor, Dr. Muhammad Azam, for his guidance, patience, and constructive feedback throughout this project. His insights at every stage, from refining the research objectives to reviewing the final results, were invaluable to the successful completion of this work.

I am also grateful to Whitecliffe, New Zealand, for the resources and academic support provided throughout my studies.

Finally, I thank my family and friends for their constant encouragement, patience, and understanding throughout this journey.

Abstract

The fast adoption of Internet of Things (IoT) technologies has raised new security concerns particularly for devices that need lightweight authentication mechanisms. One of the most persistent threats is the replay attack. A replay attack occurs when an attacker intercepts and replays an authentic message in order to gain access to another system. Because replay attacks rely on valid messages rather than forged ones, they are difficult to detect using conventional security techniques. Existing lightweight replay protection mechanisms typically rely on either nonces or counters. However, each approach has limitations. Nonce-based mechanisms become vulnerable following power loss or when memory resets, while counter-based mechanisms can be compromised through rollback or desynchronisation attacks. Although these weaknesses are well documented individually, there has been limited comparative evaluation of nonce-only, counter-only and hybrid approaches under realistic fault conditions on resource constrained IoT systems.

This project addresses that gap with a controlled, side by side comparison of nonce, counter and hybrid validation on the same constrained, oneway channel and it additionally measures the validation latency that each design adds.

This project designs and evaluates a lightweight hybrid nonce-counter framework that combines both freshness checks into a single validation step. Then it compares against nonce-only, counter-only and an unprotected baseline. This study uses a Python simulation with 1 000 seeded trials per test, four metrics like Detection Rate,Attack Success Rate, False Rejection Rate and per message validation latency and seven attack scenarios including reset, desynchronisation and counter-rollback.

The results demonstrate that while each individual mechanism fails under at least one realistic attack scenario, the proposed hybrid framework successfully detects all evaluated attacks. Its measured costs are a 1.96% false-rejection rate at the short 8-bit nonce setting and the highest validation latency of the four methods, which is still negligible against the physical door unlock time of approximately 100 milliseconds. The project delivers a working, reproducible simulation and a practical and evidence based design guideline for replay protection in a resource constrained smart car IoT systems, together with an honest statement of where lightweight one-way freshness helps and where it does not.

Keywords: Replay attack; Smart car IoT; Remote Keyless Entry (RKE); Lightweight authentication; Nonce; Rolling code counter; Hybrid validation; Freshness verification; False Rejection Rate; Simulation study.

Contents

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Background of Study . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Project Aims and Objectives . . . 2
  - 1.3.1 Aim . . . 2
  - 1.3.2 Objectives . . . 2
- 1.4 Project Questions . . . 3
- 1.5 Project Hypotheses . . . 3
- 1.6 Project Scope . . . 3
- 1.7 Significance of Project . . . 4
- 1.8 Expected Project Outcomes . . . 4
- 1.9 Structure of the Report . . . 5

2 Literature Review 6
- 2.1 Replay Attacks in IoT and Smart Car Systems . . . 6
- 2.2 Attacks on Remote Keyless Entry Systems . . . 6
- 2.3 Lightweight Authentication Techniques for Replay Prevention . . . 7
- 2.4 Detection-Based Approaches . . . 8
- 2.5 Hardware-Based and Advanced Security-Based Approaches . . . 8
- 2.6 Limitations in Existing works . . . 9
- 2.7 Summarising and Research Gap . . . 9

3 Project Methodology 10
- 3.1 Project Approach . . . 10
- 3.2 Experimental Procedure . . . 10
- 3.3 Why the Smart-Car RKE Scenario . . . 10
- 3.4 System Overview . . . 11
- 3.5 Threat Model . . . 12
  - 3.5.1 Assets protected . . . 13
  - 3.5.2 Communication environment . . . 13
  - 3.5.3 Adversary capabilities . . . 13
  - 3.5.4 Attackers limitations . . . 13
  - 3.5.5 Trust assumptions . . . 13
- 3.6 Proposed Hybrid Security Framework . . . 14
  - 3.6.1 Sender-Side Process . . . 19
  - 3.6.2 Receiver-Side Validation . . . 19
  - 3.6.3 Validation Module . . . 19
  - 3.6.4 Formal Validation Rules . . . 19
  - 3.6.5 Computational Complexity . . . 20
- 3.7 Implementing a Realistic Operating Conditions . . . 21
- 3.8 Implementation . . . 22
  - 3.8.1 Simulation Scenario . . . 22

3.8.2 Simulation Parameters . . . 23
3.8.3 Experimental Design . . . 23
3.9 Simulation Assumptions and Scope . . . 24
3.10 Implementation Components . . . 25
3.11 Implementation Challenges and How They Were Addressed . . . 26
3.12 Ethical Consideration . . . 27
3.13 Evaluation . . . 27
3.14 Evaluation Metrics Selection . . . 27
3.14.1 Detection Rate . . . 28
3.14.2 Attack Success Rate (ASR) . . . 28
3.14.3 Latency . . . 29
3.14.4 False Rejection Rate (FRR) . . . 29

4 Results and Analysis **31**
4.1 Results for Standard Replay Scenarios . . . 31
4.2 False Rejection Rate and the Nonce-Field Sweep . . . 32
4.3 Results for Robustness Scenarios . . . 33
4.4 Security Capability Summary . . . 34
4.5 Latency Analysis . . . 35
4.6 Hypothesis Evaluation . . . 36
4.7 Overall Comparison . . . 37
4.8 Summary of Findings . . . 37

5 Discussion **38**
5.1 Interpretation of the Findings . . . 38
5.2 Comparison with Previous Studies . . . 38
5.3 Novelty and Contribution . . . 39
5.4 Strengths and Weakness . . . 39
5.5 Impact on Legal, Cultural and Ethical Practices . . . 40
5.6 Implications . . . 40
5.7 Summary . . . 41

6 Conclusion **42**
6.1 Recommendations . . . 42
6.2 Limitations and Future Work . . . 42

A Appendix A: Source Code **47**
Appendix A - message_final.py . . . 47
Appendix A - sender_final.py . . . 48
Appendix A - receiver_final.py . . . 50
Appendix A - attacker_final.py . . . 53
Appendix A - main_final.py . . . 55
Appendix A – Plot_Graph.py . . . 59

Glossary **64**

List of Figures

1 System architecture for our proposed hybrid System 12
2 Validation logic for Method 1 (No Validation baseline). 15
3 Validation logic for Method 2 (Nonce-Only uniqueness check). 16
4 Validation logic for Method 3 (Counter-Only freshness check). 17
5 Validation logic for Method 4 (proposed Hybrid Nonce + Counter). . . . 18
6 Attack Success Rate across all seven attack scenarios *(lower is better)*. . . 32
7 False Rejection Rate versus nonce field length. Longer nonce, fewer wrongly rejected genuine messages. 33
8 False Rejection Rate by method at the 8-bit headline setting. Counter-Only is structurally zero. 33
9 Security capability: number of the seven attack scenarios each method neutralised (Attack Success Rate $\leq 1\%$). Nonce-Only fails the *reset* and *desync* scenarios (5/7); Counter-Only fails only the *rollback* scenario (6/7); the Hybrid neutralises all seven. 35
10 Mean per-message validation latency by method, averaged across all scenarios. 36

List of Tables

1 Expected Performance Outcomes Per Method 5
2 Threat model summary 14
3 Per-message time and memory cost of each method 20
4 Simulation Parameters 23
5 Evaluation Metrics 27
6 Results for the Standard Scenarios (1000 trials; DR/ASR identical across all four standard attacks) 31
7 False Rejection Rate versus nonce field length (clean traffic, 1000 trials) . 32
8 Results for the Reset, Desync and Rollback Robustness Scenarios (1000 trials per cell) 34
9 Consolidated results for all four methods. 37
10 Comparison with prior studies on replay protection for RKE channels . . 39

1 Introduction

1.1 Background of Study

Smart homes, industry operations intelligent transportation systems are all becoming more important as the Internet of Things (IoT) expands (Kumar et al., 2024). Smart car systems, in particular, utilize significant reliance on wireless communications to achieve features such as Remote Keyless Entry (RKE), Remote Engine Starting and Vehicle Access Control. While these services considerably improve user comfort and automate operations, they also introduce serious security risks due to the open and limited resources of the IoT communication environment.

One of the most serious risks to smart car systems is the replay attack. An attacker can intercept an authentic communication signal and retransmit it again to gain access to a targeted system. Although captured messages are generally highly encrypted and legitimate, a system may mistakenly think that a replayed message is real, making it difficult to detect as authentic. The replay attack is effective against IoT devices, because of the lack of secure freshness verification and the need for lightweight protocols. Researches has repeatedly revealed that a large percentage of IoT devices are still vulnerable to replay attacks, emphasizing the severity of the issue.(Lazzaro et al., 2024).

Replay attacks are no longer limited to reusing signals in smart automobiles. Attacks such as RollJam and RollBack demonstrate that even today's powerful rolling-code-based RKE systems are vulnerable to replay Attacks and resynchronization techniques. An example of this is the RollBack attack (Csikor et al., 2023). This attack allows an attacker to reuse previously acquired signals without having to intercept them a second time. This exposes an issue in existing replay security. As a result, the security of present solutions cannot be relied on to provide long term defense against replay assaults in a practical scenarios.

To mitigate replay attacks the lightweight authentication techniques like **nonce-based and counter-based** are commonly utilized. They are intended to keep messages fresh by preventing previously transmitted messages being resent. Each of these techniques has its own disadvantages. Nonce-based systems may result in high communication expense, whereas counter-based systems have synchronization issues and can trigger state rollback attacks (Coston et al., 2025). Recent studies has also shown that timestamp spoofing and desynchronisation can compromise different types of authentication in practical IoT systems(Byeon, 2026).

In practic, these effects are linked to how real systems are built. Real rolling code receivers dont insist on the exact next counter value and they accept any counter within a small forward "window" and hence an accidental button presses out of range do not lock the owner out and this window is what replay and resynchronise attacks abuse (Csikor et al., 2023; Microchip Technology Inc., 2011). A nonce, on the other hand, is drawn from a limited range of numbers, so on a constrained device two genuine messages can occasionally pick the same value by chance. When that happens an real message is wrongly rejected (Køien, 2015). A fair testing is needed to measure that not only how many attacks are stopped, but also how often genuine messages are wrongly rejected.

While there has been a lot of research into these mechanisms, the majority of these studies analyze them either separately or under ideal conditions. Comprehensive comparison analysis for smart car environments is constrained due to a number of issues like latency, processing limitations and unreliable connectivity. Additionally, relatively little study has been conducted on the usage of hybrid systems that integrate multiple freshness validation techniques to make up for limitations in individual validation methods.

As a result, this study will investigate the vulnerability of replay attacks in IoT environments for smart vehicles and propose a lightweight hybrid framework that combines nonce-based and counter-based validation techniques. This study will contribute to a improved understanding of how to achieve an appropriate balance between security and performance by examining both the individual and combined evaluation methodologies for a smart vehicle communication environment.

1.2 Problem Statement

Replay attacks create a major threat to the security of IoT systems in smart cars. These attacks primarily risk the security of keyless wireless communication since an attacker can intercept legitimate authentication messages and retransmit them, enabling unauthorized access. At present, a significant reason for this vulnerability is a lack of strong message freshness verification mechanisms in the smart car IoT system, which provides a means of preventing replay attacks.

There are lightweight protection to Replay attacks, such as nonce-based and counter-based schemes, however these are normally tested separately and under controlled conditions. They do not accurately represent the complexity of real-world environments.

Modern attack methods demonstrate that more sophisticated safety measures such as rolling codes, can be bypassed using replay-based resynchronization techniques. As a result, there is significant uncertainty about the efficacy of existing solutions when put into reality. This suggests that lightweight Replay protection techniques should be properly evaluated, as well as hybrid solutions developed to effectively overcome limitations.

1.3 Project Aims and Objectives

1.3.1 Aim

To design and evaluate a lightweight hybrid nonce-counter framework for replay attack prevention in smart car IoT systems.

1.3.2 Objectives

- To analyse replay attack behaviour in smart car systems.
- To implement a simulation environment representing smart car IoT communication with nonce based and counter based replay protection mechanisms.
- To design a hybrid nonce counter framework that addresses the individual limitations of nonce-based and counter based approaches.

- To evaluate the security effectiveness of nonce based, counter based and hybrid approaches against replay attacks through performance metrics including detection rate, attack success rate and latency overhead.

1.4 Project Questions

- How do replay attacks impact smart car IoT communication systems?
- What are the individual limitations of nonce-based and counter-based replay protection mechanisms when applied in a simulated smart car IoT environment?
- How does a hybrid nonce-counter framework improve replay attack resistance compared to nonce-based and counter-based approaches used individually?
- What are the performance trade-offs, in terms of detection rate and attack success rate and latency overhead between other approaches?

1.5 Project Hypotheses

Based on the analysis of nonce based and counter based authentication mechanisms, this study formulates the following hypotheses:

- **Detection Capability:** The Hybrid method will achieve a Detection Rate at least high as either the Nonce-only or Counter-only method across all replay and robust scenarios tested and higher than at least one of them.
- **Attack Resistance:** Attack Success Rate will be the lowest for the Hybrid method across all types of attack compared to the No Protection baseline (100% ASR).
- **Performance amd Usability Trade offs:** The Hybrid method will carry a measurable cost relative to the single methods. A higher validation latency, because it performs two checks instead of one and non zero False Rejection Rate inherited from the nonce check. These indicate a trade off between security and usability, rather than a cost free improvement.
- **Attack Type Vulnerability:** Nonce only methods will fail when its stored values are lost after a power-loss reset and when a genuine message is jammed in transit (desynchronisation).The Counter-only method will fail when its counter is forced backwards by a rollback attack. The Hybrid method will handle all of these.

1.6 Project Scope

This project focuses on replay attacks in smart car IoT systems, especially keyless communication. This project includes

- Simulation of communication
- Replay attack implementation
- Evaluation of nonce and counter based techniques
- Design and testing of a hybrid framework

1.7 Significance of Project

Replay attacks continue creating major challenges for IoT systems, especially those using smart cars. Currently, the solutions typically focus on detection or use complicated techniques that are impractical for lightweight IoT devices.

In this study, we provide a effective and simple approach based on a hybrid nonce counter system. Unlike prior studies, this study focuses on:

- Comparing nonce-based and counter-based techniques.
- Combining both into a hybrid framework.
- Evaluating them in a realistic simulated environment.

The key significance of this project includes:

- Providing a lightweight and efficient security solution for smart car IoT systems.
- Improving understanding of replay attack behaviour in real-world scenarios.
- Showing how a hybrid approach can improve reliability and security

This project can be useful for:

- IoT system designers.
- Automotive Security Engineers.
- Researchers working on lightweight authentication.

Overall, this study helps in designing more secure and efficient IoT communication systems.

1.8 Expected Project Outcomes

This project aims to simulate impact of replay attacks on smart car IoT systems, Provide a comparative analysis of lightweight replay protection techniques. Also,validate the effectiveness of a hybrid Nonce-Counter framework and offer practical insights for designing a secure and an efficient IoT authentication mechanism. The intended outcomes are:

- A working Python based simulation environment of a wireless authentication between a key fob and a vehicle ECU.
- A comparison of four validation configurations (no protection, nonce-only, counter-only and hybrid) against multiple replay attack scenarios, including the reset, desynchronisation and rollback scenarios, together with a study of how the nonce field length affects the false-rejection of a genuine messages.
- A practical and defensible design recommendation for replay protection in a resource constrained smart car systems.

The expected quantitative outcomes for each method are shown in Table 1.

Table 1: Expected Performance Outcomes Per Method

Method	**Detection Rate**	**Attack Success Rate**	**False Rejection**	**Latency**
No Protection	0%	100%	None	N / A
Counter-Only	High	Low	None	Low
Nonce-Only	High	Low	Low	Medium
Hybrid	Highest	Lowest	Low	Moderate

These predicted values will be validated through the simulation experiments. This study will also deliver a working Python-based simulation environment and practical design for authentication in a resource constrained smart car IoT systems.

1.9 Structure of the Report

The remainder of this report is organised as follows. Section 2 reviews the literature on replay attacks, remote keyless entry (RKE) and lightweight freshness mechanisms and identifies the gap this project addresses. Section 3 presents the methodology. The threat model, the proposed hybrid framework, the simulation design, the evaluation metrics and the ethical scope. Section 4 reports and analyses the results across the standard and robustness scenarios. Section 5 interprets the findings, compares them with prior studies and evaluates the solution's novelty, strengths, limitations and its legal, cultural, and ethical impact. Section 6 concludes with the contribution, recommendations and future work. The source code is provided in Appendix A.

2 Literature Review

The sudden growth in interest in the Internet of Things has enhanced the ability to automate and connect devices in a variety of sectors, including smart homes, healthcare providers and transportation logistics. The rapid growth of IoT devices has also increased their exposure to security threats, particularly those targeting common network vulnerabilities and communication protocols. These threat can be classified as malicious, un intentional or a combination of both. The most dangerous assault on IoT devices comes from replay attacks, which exploit vulnerabilities in either the verification procedure used to establish the freshness of communications or the authentication process used to validate identity (Albinali & Azzedin, 2023).

IoT devices have limited capabilities due to limited processor, memory and power resources. Their typical applicability is within a very confined setting. Due to these constraints, IoT systems often rely on lightweight security mechanisms that may not provide strong protection against advanced attacks. Many existing protocols for authenticating IoT devices are designed to be lightweight and efficient and dont have a strong protection against replay attacks. As a result, attackers commonly exploit methods of communication and obtain access to an IoT system without breaching the encryption.

2.1 Replay Attacks in IoT and Smart Car Systems

The replay attack takes place when an attacker intercepts and replays a valid communication, manipulating the system into assuming that the broadcast is authentic. It is one of the most serious types of attacks because there is no need to decrypt a communication and by simply replaying it can authenticate itself (Albinali & Azzedin, 2023).

Replay attacks can interrupt IoT networks or their operations by decreasing packet delivery percentages and increasing packet delivery times. Empirical studies have shown that replay attacks can reduce packet delivery ratios to less than 60% and increase communication latency up to 50%, significantly degrading the network performance(Albinali & Azzedin, 2023, p. 14). Research on routing protocols for IoT systems has shown that replaying control messages can have a major impact on network performance and cause instability.

Smart car features, such as remote keyless entry are top targets for replay attacks because they rely on radio wave communication between the key fob and the car itself giving the attacker easy access to messages that can be captured and replayed later to open car doors or start engines (Al-Haija & Alsulami, 2022).

In smart car systems, particularly Remote Keyless Entry (RKE) are highly vulnerable because they rely on wireless communication between the key fob and the vehicle. Hence making it easy for the attackers to capture and replay signals(Bianchi et al., 2025).

2.2 Attacks on Remote Keyless Entry Systems

Early Remote Keyless Entry (RKE) systems used static code, which left them in a vulnerable position when intercepted and a repeating signal may be exploited multiple times. Rolling code techniques were developed to address this issue by generating new code with each transmission. A new code is delivered each time a user transmits as determined by an updated counter.

Some recent studies show that rolling code systems are still vulnerable to advanced replay based attacks due to weaknesses in the synchronization mechanisms and the protocol design (Bianchi et al., 2025).

The synchronisation behaviour of these attack targets is well documented. The widely used KeeLoq rolling-code design specifies a forward acceptance window of codes, so that out of range button presses do not desynchronise the fob and this window is precisely what jam and replay attacks rely on (Eisenbarth et al., 2008; Microchip Technology Inc., 2011; Zhang & Tsou, 2012).

However, studies indicates that rolling code systems are not secure. Rolling code systems are vulnerable to advanced attacks since both the key fob and the car use insecure synchronization protocols. The RollBack attack is an example that can force a system to revert to a previous state, allowing the attacker to use a previously acquired signal, especially since the attack has no time component and consequently has a high degree of practicality (Csikor et al., 2023).This demonstrates that even modern RKE systems cannot fully prevent replay attacks and require improved freshness validation mechanisms.

Furthermore, modern Passive Keyless Entry (PKE) systems have drawbacks in their cryptographic architecture like bad protocol implementations, lack of mutual authentication and inadequate protocol designs. These vulnerabilities makes today's PKE systems vulnerable to replay and relay attacks (Wouters et al., 2019).

2.3 Lightweight Authentication Techniques for Replay Prevention

Due to resource constraints in Internet of Things (IoT) devices, a number of lightweight and simple techniques have been developed for preventing replay attacks with low computational power.

These techniques are called lightweight because each one reduces replay detection to a single check that needs no heavy cryptography. A nonce check is one lookup in a list of recently seen values and a counter check is one comparison of two numbers. A timestamp check is one subtraction against a clock. None of them requires public key operations or a handshake or a negotiated session key, which is what makes them suitable for the small, battery powered microcontrollers used in devices such as key fobs (El-Hajj et al., 2019; Kumar et al., 2024)

Nonce methods are well-known examples of lightweight authentication. Nonce methods use a randomly generated value that is unique to the session in which it was created, making sure the message is legitimate when it is first received by the receiver (Køien, 2015; Menezes et al., 1996).

Counter methods are another widely used method to deliver lightweight authentication. Counter methods rely on sequencing to determine whether a received message is a duplicate of one that has already been received (3GPP, 2020; Menezes et al., 1996; Zhang & Tsou, 2012).

Timestamp methods are another example of lightweight authentication. These methods provide a way for authenticating messages within a specified "window" of time after they are created. However, they depend heavily on clock synchronization across IoT devices (Menezes et al., 1996).

Hybrid methods are another explored technique by combining multiple mechanisms such as nonce, counter and timestamp to improve replay protection.For example, modern IoT protocols integrate counters with timestamp-based validation to overcome the limitations of individual approaches(Krentz & Voigt, 2024).

It is important to note that combining a sequence of numbers with a random value is not a new primitive concept. The Handbook of Applied Cryptography cites random numbers, sequence numbers and timestamps as the three common "time-variant parameters" for freshness and notes that they can be combined (Menezes et al., 1996). The UMTS/LTE authentication standard already pairs a fresh sequence number with a random challenge (3GPP, 2020). The contribution of this project therefore is not the idea of combining the two, but an actual evaluation of the security and usability trade offs of doing on a limited, oneway RKE channel.

Additional research started to investigate the use of **Session Key** approaches for lightweight authentication. Each communication session generates its own dynamic key also known as a session key. This significantly improves security because intercepted messages cannot be utilized in future communication sessions (Rakshit et al., 2025).

2.4 Detection-Based Approaches

In addition to preventative strategies, some research focuses on detecting replay attacks using machine learning and signal analysis approaches. Recent studies used Software Defined Radio (SDR) and machine learning models to detect replay attacks in smart car systems with high accuracy, demonstrating detection rates above 90% (Quintero et al., 2023).

For example, deep learning models have been used to precisely differentiate authentic and fraudulent signals in keyless automobile systems.

Using a pre-trained neural network model, one such solution detected replayed signals with more than 99 % accuracy (Al-Haija & Alsulami, 2022). While these solutions work, they are computationally intensive and may not be appropriate for low-power IoT devices.

2.5 Hardware-Based and Advanced Security-Based Approaches

Physical Unclonable Functions (PUFs) are a hardware-based solution to improving authentication security by creating unique identities for each device. In addition, PUFs give a high level of security, but they require additional hardware and increase the overall complexity of the system(Alharthi & Altuwaijri, 2025).

Another advanced solution is **Block Chain** based on an authentication scheme that authenticates the device communication using blockchain technology and a distributed ledger. This method boosts device trust and security, but it also introduces latency and computational overhead, making it unsuitable for real time IoT applications (Djinko et al., 2022).

An additional technique approach is a two way attack-response authentication, which ensures that a recorded message is never reused. Recent PUF-based mutual authentication schemes follow this route and are reported to defend against replay, RollJam and even the RollBack attack, at the cost of requiring a return channel and additional hardware. (Parameswarath & Sikdar, 2022, 2024).

2.6 Limitations in Existing works

Despite the wide range of proposed solutions there are several limitations remain in current project:

- Most approaches focus on a single technique instead of comparing multiple methods.
- High-security solutions often ignores resource constraint.
- Existing replay protection mechanisms fails under real world conditions such as desynchronization, delay attacks and rollback attacks.
- There is a limited evaluation of the hybrid approaches that combine multiple freshness mechanism in IoT environments.
- The usability cost of these mechanisms, especialy how often a genuine message is wrongly rejected, is rarely measured alongside with their security.

2.7 Summarising and Research Gap

The Literature review finds a clear gap in the Project studies. Present studies does not provide direct comparisons between lightweight replay attack prevention techniques, specifically nonce based replay attack prevention techniques versus counter based replay attack prevention techniques, to better understand how well they both work for smart car IoT systems.

There has also been a lack of practical implementations and performance assessments of both methodologies in practical operating contexts. Existing researches has focused solely on the theoretical models or high level simulations of both methodologies and failing to include any of the real world constraints associated with their use like latency and synchronization concerns and limited available resources.

Despite the wide adoption of nonce-based and rolling-code systems, there is very little experimental literature comparing these two types of methods across multiple replay resilience scenarios (reset, desynchronization, rollback and the usability trade-off resulting from false rejection) on a particular constrained one-way channel. This study will provide a direct answer to this question.

As a result, the objective of this project is to design and evaluate a hybrid nonce and counter framework and compare it with existing individual approaches to identify a more secure and efficient solution for smart car IoT systems.

3 Project Methodology

This study follows a simple experimental approach to analyse replay attacks and then evaluate lightweight prevention techniques in a smart car IoT system. This study follows a simulation based experimental approach to analyse replay attacks and evaluate lightweight prevention techniques in a smart car IoT system under controlled conditions. The entire system is simulated using a software based environment, which allows controlled testing without using real hardware.

3.1 Project Approach

The project uses simulation and comparison method. The project is based on simulation and comparative analysis of the multiple replay attack prevention techniques. A basic communication system is created in order to represent a smart car environment. The system is then tested under normal conditions, attack conditions and under protected conditions.

3.2 Experimental Procedure

Each of the seven attack scenarios was executed for $N = 1000$ independent trials, giving $7 \times 1000 = 7000$ trials per validation method. Within every trial, a fresh key fob, ECU and attacker were instantiated and the identical seeded message stream (global seed = 42) was validated by all four methods.

This paired design ensures that any difference between methods arises from the method itself and not from variation in the message stream. Each reported metric is the mean over the 1000 trials of its scenario, with 95% confidence intervals. The full experiment therefore comprises 4 methods $\times$ 7 scenarios $\times$ 1000 trials = 28,000 method–scenario evaluations.

3.3 Why the Smart-Car RKE Scenario

While replay protection is a typical concern in IoT, the Remote Keyless Entry (RKE) feature of a smart car is particularly challenging example. The three characteristics that differentiate RKE from an average IoT sensor network are as follows.

First, the limitation is that fobs are *broadcast (one-way)* and do not get feedback from the car. Connected IoT devices don't usually support rich two-way handshakes such as challenge-response authentication, mutual authentication and session key negotiation. The defense is limited to what can fit into a single message (outbound). Therefore, using a nonce or counter meets that requirement.

Second, Another restriction of fobs is their limited resource capacity. Fobs can run for years on a coin-cell battery, but they can't use big nonce fields, complex cryptography or maintain large permanent states. The final result is a short nonce field, which accounts for the false-rejection cost measured in this work, as well as the use of PUF and blockchain based methods(Alharthi & Altuwaijri, 2025; Djinko et al., 2022).

Third, both the asset targeted by the attacker and the instruments used by the attacker are real and serious.When an attacker successfully executes a replay, they do not modify a data transmission. The tools used by attackers include commodity goods such as software-defined radios and with low-cost devices such as Flipper Zero. Recording and replaying data is simple for someone without the necessary skills (Bianchi et al., 2025; Kamkar, 2015) or understanding to carry out such an attack. Additionally, established methods like RollBack attacks can bypass even modern rolling code systems (Albinali & Azzedin, 2023). The combination of a one direction channel, a small amount of energy necessary to launch an attack and a very valuable item that incurs an inexpensive attack demonstrate that RKE is a far more difficult test of lightweight replay security than a generic IoT implementation.

3.4 System Overview

The system includes three main components:

- A Sender (Key Fob) : builds each message as the tuple $\langle Command,\ N,\ C \rangle$, where N is a fresh nonce and C an incrementing counter, and transmits it over the open RF channel.
- A Receiver (Car) : runs the Validation Module on every received message and either actuates the command (lock, unlock, start) or rejects it.
- An Attacker (Captures And Transfers Reply Signals) : sits on the wireless channel and can capture, store and replay legitimate transmissions and in the rollback case force the receiver's counter backwards.

Figure 1 shows how these components are arranged. The key fob and the vehicle ECU sit at opposite ends of a single broadcast channel and the attacker captures *on* that channel. Every messages the fob sends is also visible to the attacker, who may store and replay it later. The receiver's Validation Module is the one that decides whether an incoming command is fresh before it is allowed to reach the actuator. The protected asset is the accepted vehicle command and the full attack model (capabilities, limitations and trust assumptions) is detailed in Section 3.5.

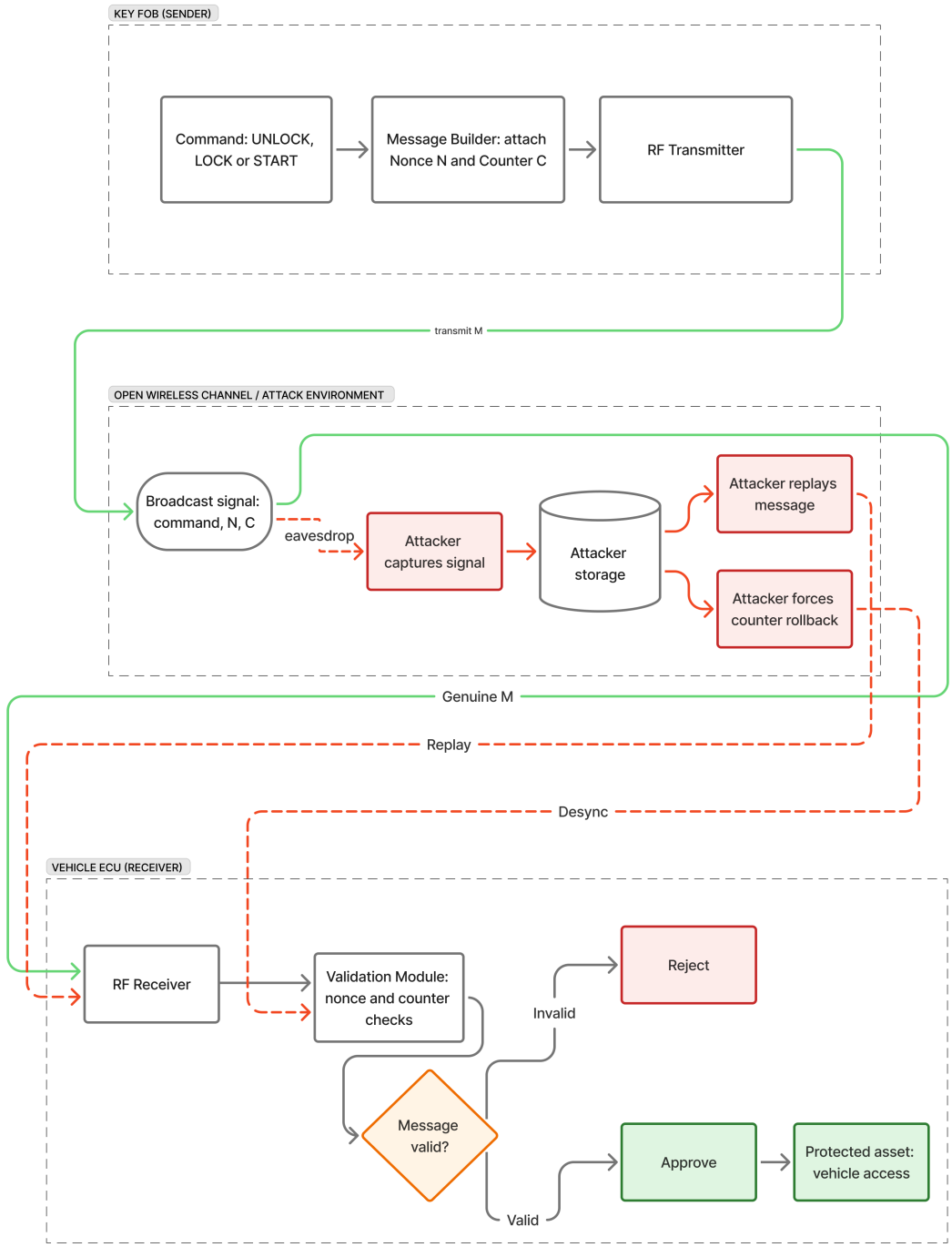

Figure 1: System architecture for our proposed hybrid System

3.5 Threat Model

This section describes the adversary, protected assets, communication environment and trust assumptions. Following results can be compared to a single explicit model, rather than assumptions made by the reader.

3.5.1 Assets protected

The secured item that must be protected is access and control of the vehicle. That is the ability to send a command to the ECU (for example, a `UNLOCK`, `LOCK` or `START` command), as well as ensuring that the ECU has accepted the command by confirming that it was sent only once recently from the fob in question and does not reuse an old command. As mentioned in the literature, confidentiality of command contents is not a security aim because efficient replay attacks are achievable against all types of encrypted messages in which an attacker reuses the same valid ciphertext without decrypting it(Lazzaro et al., 2024).

3.5.2 Communication environment

The Commands travel over a single, oneway (broadcast) message stream. The key fob transmits a command to the ECU and receives nothing back. There is no return channel from the ECU to the fob in the baseline model, interactive defences such as challenge and response are unavailable and one-way freshness tokens (a nonce or counter carried in the outgoing message) are the only option for ensuring that a delivered command is fresh.

3.5.3 Adversary capabilities

The attacker is represented in the strongest reasonable form, resulting in worst case scenarios for the defender.

- **Perfect capture** The attacker records every transmitted message without any loss. This eliminates the challenge of interception and focuses on the validation logic's effectiveness. (Csikor et al., 2023).
- **Jamming or selective blocking.** The attacker can prevent any selected message from reaching the ECU while still recording it (`silent_capture`), which is the precondition for the desynchronisation and RollJam-style attacks. (Kamkar, 2015).
- **Replay in any order and multiplicity.** Captured messages can be resent once, many times, in reverse order or selectively.
- **Rollback** In the rollback scenario, attacker can force the ECUs stored counter backward while leaving nonce memory intact, reflecting the documented RollBack attack (Albinali & Azzedin, 2023).

3.5.4 Attackers limitations

The attacker cannot break the underlying cryptography or Create an valid message from scratch or recover secret keys. Adaptive or learning based attackers are out of scope. The attacker follows a fixed strategy per scenario.

3.5.5 Trust assumptions

The fob and ECU are expected to have a previously established trust relationship that was formed out of band before deployment. The ECU is not physically tampered with, except for the modeled power loss (reset) and counter rollback. The fob is considered as genuine, not cloned.

A consolidated view of this model is given in Table 2.

Table 2: Threat model summary

Dimension	**This work**
Protected asset	Accepted vehicle command (unlock / lock / start), message freshness
Security goal	Freshness only, not confidentiality or integrity of content
Channel	One-way, fob → ECU no return channel, broadcast RF
Attacker can do	Perfect capture, jam and block a chosen message, replay in any order, force counter rollback
Attacker cannot do	Break cryptography, forge or alter a valid message, recover keys, adaptive and learn
Trust assumptions	Pre-shared fob and ECU pairing, trusted, untampered ECU and fob

3.6 Proposed Hybrid Security Framework

The proposed framework of constructing a hybrid replay attack prevention system integrates nonce-based and counter-based authentication methods into a single system. This enables for the integration of the best aspects of each technique, resulting in a lightweight authentication mechanism suitable for implementation in IoT environments such as smart car systems.

When using IoT communication methods, the freshness of the message (that it is transmitted in order and only once) and the sequencing of the messaging (that it is received correctly) are essential factors to consider. Using a nonce-based method makes sure each transmitted message is different from other transmitted messages. By using a counter-based method, it assures that all sent messages are received in the same order in which they were sent. However, when employed alone these has limitations. This study provides a solution on developing a hybrid system that combines both methodologies into a single system.

The validation logic of each method is presented in four separate figures (Figures 2–5). Each method is evaluated independently on the same incoming message, so the figures are deliberately not drawn as one connected sequence. The figures are placed alongside the description of the method they belong to.

No Protection (Method 1) serves as the baseline control case as it is accepting all messages without performing any validation checks. This establishes a scenario where Attack Success Rate is 100%.

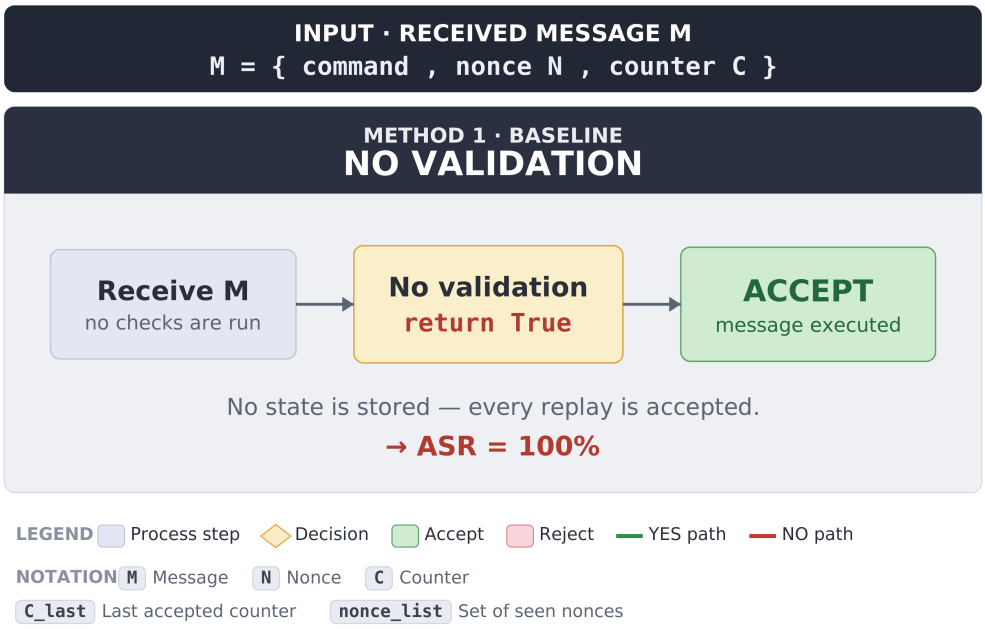

Figure 2: Validation logic for Method 1 (No Validation baseline).

Nonce-Only validation (Method 2) checks whether the received nonce has been seen previously by searching the stored nonce list. If a replay attack is detected, the message is immediately rejected. However, if the nonce is fresh, it is stored and the message is accepted. This approach eliminates delayed and multiple replay attacks, but it fails when the device resets and the nonce list is deleted. It also has a usability cost: because the nonce is taken from a narrow range of numbers, two authentic messages may randomly pick the same value and the second genuine message is therefore incorrectly rejected. In this study, the False Rejection Rate is used to measure the incorrect rejection of a valid message.

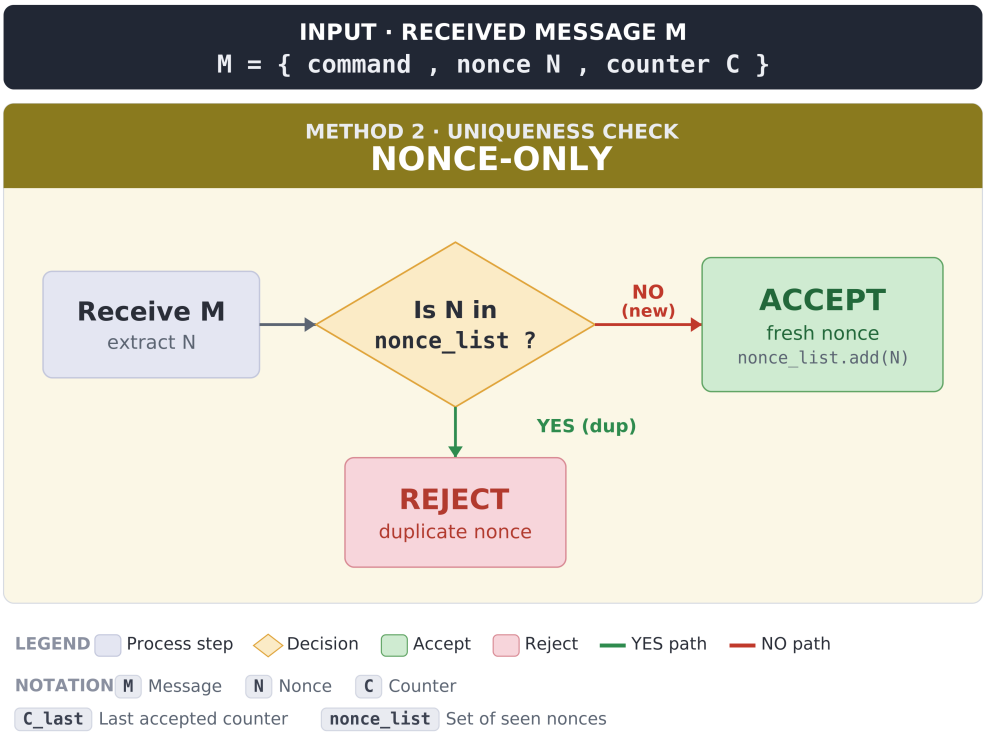

Figure 3: Validation logic for Method 2 (Nonce-Only uniqueness check).

Counter-Only validation (Method 3) checks the received counter against the last accepted counter. To simulate a true rolling-code systems, the receiver takes any counter that is bigger than the last accepted value but no more than a specified amount beyond (acceptance window), making sure that a few incorrectly entered button clicks do not lock the owner out. Counters that are equal to or less than the last acceptable value are considered replays. This method prevents delayed and multiple replay attacks, but it has two flaws. It can incorrectly reject genuine messages during desynchronization and more importantly, it is defeated by a rollback attack that forces the stored counter backwards, making the old captured codes look like new again.

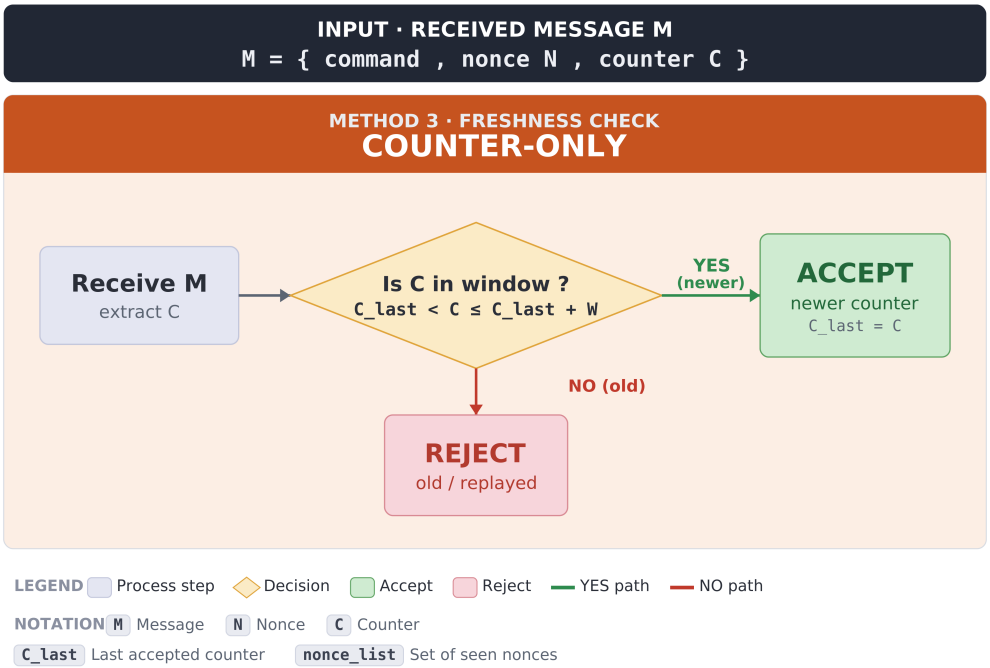

Figure 4: Validation logic for Method 3 (Counter-Only freshness check).

Hybrid validation (Method 4) implements an two step sequential check. First, it validates nonce uniqueness and only if the nonce is new it does processing and then continues to the second check. Second, it validates counter freshness and only if the counter is increasing and then the messages are accepted. This two checking approach make sure that replay attacks fail if atleast one check fails and by this we can achieve comprehensive protection against all attack types while maintaining efficiency through early rejection.

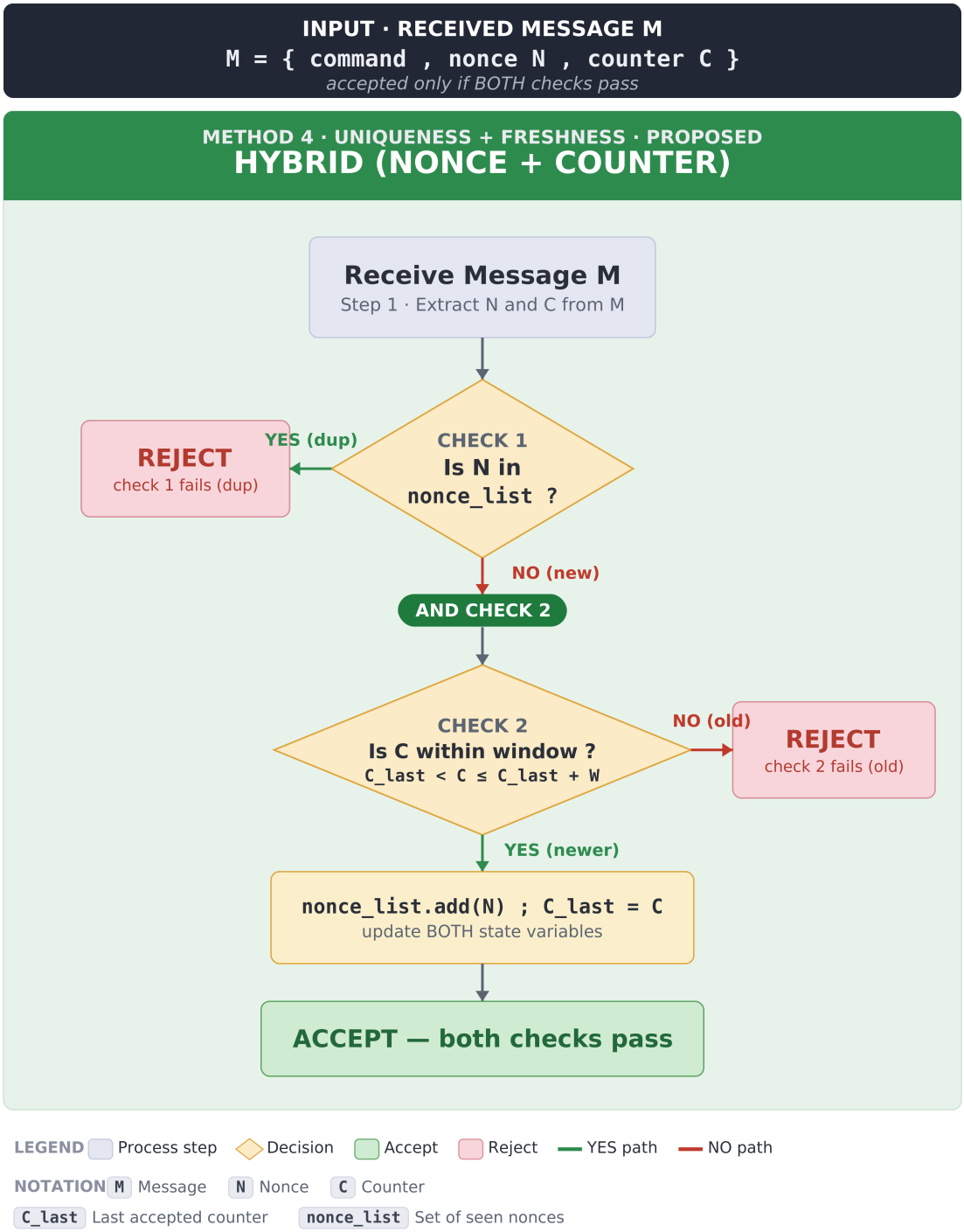

Figure 5: Validation logic for Method 4 (proposed Hybrid Nonce + Counter).

Each message generated by the sender consists of three components:

- **Command** (e.g., UNLOCK, LOCK, START)
- **Nonce (N)** - a randomly generated value unique per message
- **Counter (C)** - an incrementing sequence number

3.6.1 Sender-Side Process

The transmitter (key fob) sends data in a simple sequence. First, a random nonce value is generated to ensure that it is unique. Then counter value is increased to keep the sequence in order. These values are combined with the command to generate the final message.

3.6.2 Receiver-Side Validation

The validation module is used by the receiver (smart car system) to verify the incoming message. This module performs two levels of tests to verify freshness and sequence correctness.

3.6.3 Validation Module

Counter validation makes sure that messages receive in the correct order by saving the last successfully received counter value and comparing it to any freshly received counter values and Nonce validation makes sure messages are new to the system and have not been used earlier. Therefore, previously submitted nonce values are stored by the system for comparison with incoming nonce values. After both validations, an final decision is made. This dual validation improves reliability compared to using a single method.

3.6.4 Formal Validation Rules

Let an incoming message be $m = \langle \mathit{Command},\, N,\, C \rangle$,where N is a nonce drawn from a 8-bit field and C is the message counter. The receiver stores the set $\mathcal{N}$ of nonces it has already accepted and the last accepted counter C_{last}, and uses a fixed forward acceptance window W.

No validation (baseline): Every message is accepted:

$$\text{accept}(m) = \text{True}. \tag{1}$$

Nonce-only. A message is accepted only if its nonce has not been seen before:

$$\text{accept}(m) = \begin{cases} \text{True}, & N \notin \mathcal{N} \\ \text{False}, & N \in \mathcal{N}. \end{cases} \tag{2}$$

On acceptance the nonce is stored, $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$. Because N comes from a finite b-bit field, two genuine messages can collide. For a stored set of size k the probability that a fresh genuine nonce is wrongly rejected is

$$\Pr[\text{false reject}] = \frac{k}{2^b}, \tag{3}$$

Counter-only. A message is accepted only if its counter falls inside the forward window:

$$\text{accept}(m) = \begin{cases} \text{True}, & C_{\text{last}} < C \le C_{\text{last}} + W \\ \text{False}, & \text{otherwise}. \end{cases} \tag{4}$$

On acceptance the counter is advanced, $C_{\text{last}} \leftarrow C$. A counter at or below C_{last} is rejected as a replay, and a forced rollback of C_{last} makes previously captured codes acceptable again.

Hybrid. Acceptance requires both checks, with the nonce gate evaluated first:

$$\text{accept}(m) = \big(N \notin \mathcal{N}\big) \ \wedge \ \big(C_{\text{last}} < C \leq C_{\text{last}} + W\big). \tag{5}$$

Evaluating the nonce gate first gives early rejection: a replayed nonce is discarded before the counter comparison is reached.

3.6.5 Computational Complexity

Each method is also compared by how much work it does for every message and how much memory it needs. The cost is described using big-O notation, where n is the number of nonces the receiver is currently storing. A summary is given in Table 3.

Table 3: Per-message time and memory cost of each method

Method	**Time per message**	**Memory**
No Validation	$O(1)$	$O(1)$
Counter-Only	$O(1)$	$O(1)$
Nonce-Only	$O(1)$ avg.	$O(n)$
Hybrid	$O(1)$ avg.	$O(n)$

(n = number of stored nonces).

In simple terms, all four methods check a message in constant time written as $O(1)$. A counter check is a single comparison of two numbers. A nonce check is a membership test in a hash-based set of the nonces seen so far. Because the nonces are stored in a hash set, both the lookup and the insertion take constant time on average. This constant time cost depends on that choice of data structure. If the nonces were instead kept in an ordinary list and scanned one by one, each lookup would grow with the number of stored nonces, that is $O(n)$. The implementation uses Python's built-in `set`, so the average-case $O(1)$ cost applies here. The difference is in memory. Counter only has to remember one number that is the last accepted counter so its memory cost stays the same no matter how many messages arrive ($O(1)$). Nonce-only and the hybrid has to remember every nonce they have accepted, so the memory they use grows with the number of messages ($O(n)$).

The hybrid therefore keeps the constant time speed of the counter method and only inherits the growing memory of the nonce method. It means that it adds no extra cost of its own beyond running the two simple checks together. This is why the hybrid is still considered lightweight even though it combines two mechanisms. The work per message is one set lookup and one number comparison, both constant time and the only thing that grows is the set of stored nonces that the nonce method already needs.

3.7 Implementing a Realistic Operating Conditions

An early version of this simulation generated perfect results for all protected methods (100% detection and 0% attack success). That result was unrealistic and to make the results more realistic and to highlight the true differences between the methodologies, three effects obtained from how real systems operate were included to the model.

To save battery life and bandwidth, a real low power key fob generates a random nonce from a limited set of numbers. When two values are picked at random from a small range, they will eventually match by pure chance; this is the known as "birthday" effect, that involves the two values occur much sooner than expected (Køien, 2015). The similar impact is noted in cryptographic standards, which limit the number of random values that can be used under a single key in order to reduce the likelihood of a collision (Dworkin, 2007). In a freshness check that rejects any nonce it has seen previously, a chance match between two *genuine* messages cannot be distinguished from a replay, thus the genuine message is incorrectly rejected (Zenner, 2009). As a result, this study utilizes a small and customizable nonce field (an 8-bit field) to detect and quantify false rejections. Reporting both wrongly-accepted attacks and wrongly-rejected legitimate messages follows the false-accept/false-reject framework widely used in biometric systems (Jain et al., 2004).

Real rolling-code receivers accept any counter within a moving forward window, not only the exact next value in order to avoid accidental pushes from desynchronizing the fob (keeloq2011, eisenbarth2008).The simulation simulates this using a fixed window. Most importantly, this window does not weaken protection against ordinary replays. For example, old captured code still carries a counter below or close to the last accepted value and is still rejected. This is included for accuracy and because it is the necessary condition that makes the rollback attack meaningful (Bianchi et al., 2025; Csikor et al., 2023).

The two freshness values exist in different memory systems, which is what gives them complementary weaknesses:

- In a **reset** (power loss), the list of seen nonces, held in volatile memory, is deleted and the counter which is held in a non-volatile memory survives. The nonce-only method then treats every old message as new and fails but the surviving counter still protects the counter-only and hybrid methods. (Zenner, 2009).

- In a **rollback** attack, the opposite happens. The counter is forced backwards while the nonce memory is untouched. The counter-only method then accepts the old values and fails but the nonce memory still protects the nonce-only and hybrid methods (Csikor et al., 2023).

Because these two defects impact opposing sections of the design, no single freshness value can withstand both, however the hybrid that combines both wins. The hybrid framework's logic is based on this complimentary behavior rather than a raw advantage over traditional attacks.

3.8 Implementation

3.8.1 Simulation Scenario

This study uses a controlled software based simulation that simulates the wireless authentication between a key fob and a smart car's electronic control unit (ECU).The simulation focuses on testing replay protection techniques at the protocol level, rather than the physical or network layers. This project topic aims to compare the effectiveness of nonce-based, counter-based and hybrid validation logic, regardless of the transmission channel.This reflects the established Remote Keyless Entry (RKE) threat model which an attacker captures and replays a legitimate key fob signals to gain unauthorised vehicle access (Bianchi et al., 2025; Csikor et al., 2023).

A simulation-based approach was specifically selected as the method for evaluating security because it utilizes a controlled and reproducible environment, which avoids any legal or safety issues, as well as any hardware issues that would come up when evaluating replay attacks against actual vehicles. As the goal of this project is to compare four different validation methods using the same exact configuration, a software model that holds every other variable constant except for the validation method isolates the effect of the validation method used, which is better compared to a physical test setup that is subject to RF noise, timing issues specific to the device being tested, or any uncontrolled interference.The use of solely mathematical modeling was ruled out because false rejection behaviour and the resulting rollback and reset failure modes are emerging aspects of executing code and memory state and cannot be defined as a closed form.

The simulated environment consists of three logical entities:

- **Key Fob (Sender):** Generates authenticated commands (UNLOCK, LOCK, START) and transmits them to the car ECU. Each message contains a command and a randomly generated nonce and an increasing counter.
- **Car ECU (Receiver):** Receives incoming messages and validates them using one of four configurations: No protection, nonce only, counter only or the proposed hybrid framework.
- **Attacker:** Captures all transmitted messages and attempts to replay them using four different replay strategies (delayed replay, multiple replay, out-of-order replay and counter-skip replay).

Communication between the entities is modelled as a message passing within a single Python program. This allows precise control over message ordering, attack timing and validation outcomes, as well as evaluating the actual processing latency caused by each security method.

The simulation runs multiple separate sessions for analytical accuracy. Each session equates to a specific. interaction between the key fob and the car, such as unlocking and starting. During each session, the attacker intercepts legitimate messages and attempts to replay them against the ECU. The experiment is performed for all four validation conditions to obtain comparative results.

3.8.2 Simulation Parameters

Table 4: Simulation Parameters

Parameter	**Value**
Simulation entities	1 sender, 1 receiver, 1 attacker
Communication direction	Unidirectional (Key Fob → ECU)
Communication model	In-process message passing
Logical protocol	Single application-layer message format
Command set	UNLOCK, LOCK, START
Validation methods compared	None, Nonce, Counter, Hybrid
Attack types	Delayed, Multiple, Out-of-order, Counter-skip, plus Reset, Desync and Rollback scenarios
Normal messages per trial	10
Nonce field	Configurable n-bit field; 8-bit for headline results (values 0-255); swept 6 - 16 bits
Counter range	Increasing from 1; forward acceptance window $W = 256$
Trial design	Paired: the same seeded message stream is given to all four methods in each trial
Random seed	42 (fixed, for full reproducibility)
Results reported as	Mean with 95% confidence interval
Latency measurement	`time.perf_counter()`, μs precision
Implementation language	Python 3.x
Test machine - CPU	Intel Core i9-9900K @ 3.60 GHz, 8 cores / 16 threads
Test machine - RAM	32 GB
Test machine - OS	Microsoft Windows 11 Pro

All simulation parameters are fixed and stated explicitly so that the experiment is fully reproducible. The values are summarised in Table 4.

3.8.3 Experimental Design

The experiment follows a controlled design. The objective is to test each validation method on different attack type, so that any difference in outcome can be linked to the method rather than to a confounding factor.The paired, repeated-trial design with fixed seeding and confidence-interval reporting follows established guidance on controlled experimentation and statistical analysis in software engineering research (Arcuri & Briand, 2014; Wohlin et al., 2012).

Design structure: The experiment is a grid of (validation method × attack scenario), where each cell is one isolated experiment. The four validation methods are tested against each scenario independently. The comparison is expanded by three robustness scenarios: reset, desynchronisation and rollback.

Paired design: Because the nonce is random, results vary slightly from run to run. To make the comparison fair, each trial uses one randomly generated stream of messages and that same stream is given to all four methods. Any difference between methods in a

trial is therefore caused by the method, not by one method happening to receive luckier random values. Each trial uses a different seed derived from a fixed base seed, so the whole experiment reproduces exactly when repeted.

Isolation: Each test uses a fresh sender, receiver and attacker. The nonce list and counter are reset at the start of every test, making sure that the result in one test cannot influence another test.

Procedure for each Test:

1. The sender transmits a set of legitimate messages. The attacker captures a copy of each and forwards the original to the ECU, which validates and accepts them.
2. The attacker executes the replay strategy for that scenario (for example, replaying a single captured message, or replaying all captured messages in reverse order).
3. The ECU validates each replayed message using the method under test. Each outcome is recorded as accepted (attack succeeded) or rejected (attack detected).
4. Processing latency is measured for every message, legitimate and replayed and stored.

Repetition: Each test is repeated 1000 times. Detection rate, attack success rate, false rejection rate and latency are calculated across these trials and reported as averages with 95% confidence intervals.

Recording: Results for every tests are written to a CSV file with one row per (method, scenario) pair, recording the method, scenario, counts of legitimate and attack messages accepted and rejected, detection rate, attack success rate, false rejection rate and the latency.

3.9 Simulation Assumptions and Scope

The attacker and the trust assumptions are stated in the Threat Model (Section 3.5) and are not repeated in here. This section lists the assumptions specific to the *simulation model* , what is abstracted away, what is out of scope and which parameters are fixed. So that the boundaries of the results are explicit.

- **Communication abstraction:** The simulation models message level communication only. Physical-layer behaviour such as RF signal strength, jamming and propagation delay is out of scope.
- **Attacker capability:** The attacker is assumed to have perfect message capture ability. This represents a worst case scenario and isolates the strength of the validation mechanism from the difficulty of interception.
- **Memory availability:** The car ECU is assumed to have sufficient memory to store nonce values for the duration of a session.
- **Single communication type:** All messages use the same logical protocol. Multi protocol environments are out of scope.

- **No cryptographic encryption:** The work focuses on replay protection, which is different from message confidentiality. Encryption layers are assumed in real systems, but not simulated here as replay attacks can still occur on encrypted messages.
- **Deterministic attacker behaviour:** The attacker follows a fixed strategy per attack type. Adaptive or learning based attackers are out of scope.
- **Counter acceptance window:** The receiver accepts counters within a fixed forward window ($W = 256$), reflecting the usability tolerance of real rolling-code systems.
- **Finite nonce field:** Nonces are drawn from a fixed-width field (8-bit for headline results), so genuine collisions are possible and are measured as false rejections.

3.10 Implementation Components

The simulation is implemented across four Python modules:

- `message_final.py` — Defines the message structure. Each message object holds a command (UNLOCK/LOCK/START), a nonce (random six-digit integer between 100,000-999,999) and a counter (integer).
- `sender_final.py` — Implements the key fob. On each call, it constructs a message with a freshly generated nonce and an incremented counter and then transmits it.
- `receiver_final.py` — Implements the car ECU. It provides the four validation modes (no validation, nonce, counter, hybrid) with the counter accepted inside a forward window. It also models two robustness events:

  `volatile_reset()` (a power loss that clears the nonce list but keeps the counter) and `rollback()` (forces the counter backwards but keeps the nonce list).
- `attacker_final.py` — Implements the Attacker. It captures every transmitted message and provides the replay strategies (delayed, multiple, out-of-order, counter-skip).
- `main_final.py` — This is the main experiment controller. It builds the (method $\times$ scenario) grid over all seven scenarios, runs each test over **1000** trials,executes the relevant attack, measures latency and computes detection rate, attack success rate, false rejection rate and their confidence intervals. It also runs the nonce-field sweep and the statistical comparisons and writes the results to CSV files.
- `plot_results_final.py` — Reads the CSV files and generates the result figures (attack success rate across scenarios, false rejection rate versus nonce length, false rejection rate by method, latency, and a security capability summary).

Results are exported to a CSV file for analysis and later it is visualised using bar charts showing detection rate, attack success rate and latency comparisons across the four validation methods. The implementation source code is maintained in a Git repository to support transparency and reproducibility.

3.11 Implementation Challenges and How They Were Addressed

Several challenges arosed during development. Each is described below with the solutions that resolved it, because how these were handled is part of the contribution.

Unrealistic deterministic results. The first working version of the simulation produced only perfect outcomes. Every protected method scored 100% detection and 0% attack success. Increasing the number of trials did not change this. The resolution was structural rather than statistical. A finite, 8-bit nonce field was introduced so that two genuine nonce messages can occasionally collide by chance and produce a measurable false rejection rate and reflecting the birthday bound behaviour documented for short random fields (Dworkin, 2007; Køien, 2015; Zenner, 2009). A fixed forward acceptance window were added to the counter like mirroring real rolling-code receivers (Eisenbarth et al., 2008; Microchip Technology Inc., 2011; Zhang & Tsou, 2012). These mechanisms moved the simulation from an idealised model to one that exposes the genuine, non-binary cost of each design.

Making the rollback scenario meaningful. A counter check rejects any value that is not strictly ahead of the last accepted one, which would make a naive rollback trivially detectable. Representing rollback faithfully therefore required the forward acceptance window described above: only with a realistic window does forcing the counter backwards make previously captured codes acceptable again, which is the condition under which the documented RollBack attack operates (Csikor et al., 2023). Designing the three fault scenarios to act on different memory regions, reset and desync on the volatile nonce store, rollback on the non-volatile counter was what allowed the complementary failure pattern to emerge cleanly.

No suitable real-world dataset. An early option was to evaluate against a public dataset. The HCRL Car-Hacking dataset was assessed and deliberately rejected: it captures in-vehicle CAN-bus traffic rather than wireless RKE and contains no replay samples, so it matches neither the protocol nor the threat model of this study. Synthetic generation was chosen instead, which allowed the exact RKE message structure and the precise replay and fault behaviours to be modelled while keeping the work fully reproducible under a fixed seed.

3.12 Ethical Consideration

This project is entirely performed in a simulated environment, therefore there are no human participants, real experimental data or actual car systems involved. As a result, this investigation involves only minimal ethical risk.

The goal of this project is to analyze specific techniques implemented for replay attacks and then develop a defensive strategy for study purpose. As a result, the findings of this study will not be implemented in real-world situations or in ways that cause harm to any individuals or entities using genuine systems.

The data used in this study includes both simulated and published project data that is publicly available, thus no private or sensitive information is collected or used.

3.13 Evaluation

To evaluate the system, the following metrics are used:

Table 5: Evaluation Metrics

Metric	**Description**
Replay Detection Rate	Percentage of replay attacks successfully detected by the system
Attack Success Rate	Percentage of replay attacks that successfully bypass the validation mechanism and accepted as legitimate messages.
False Rejection Rate	Percentage of *genuine* messages that are wrongly rejected by the validation mechanism.
Latency	Time taken to process and validate each message

3.14 Evaluation Metrics Selection

The hybrid architecture will be evaluated using performance indicators, which have been employed to assess security efficiency, system flaws, usability and processing overhead. These evaluations were selected from prior studies in the field of replay attack studies as well as lightweight authentication approaches for IoT devices, in order to meet accepted scientific criteria in the evaluation area.

The four selected metrics are:

1. **Detection Rate (DR)** - measures security effectiveness
2. **Attack Success Rate (ASR)** - measures system vulnerability
3. **False Rejection Rate (FRR)** - measures usability cost
4. **Latency Overhead (LO)** - measures performance cost

3.14.1 Detection Rate

The Detection Rate is one of the most commonly used metrics to determine the performance of a method. For detecting a replay attack ,it provides a direct measure of how well the method successfully identifies malicious replayed messages that could compromise the integrity of the communication between the devices communicating over the network.

$$DR = \frac{D_{\text{detected}}}{N_{\text{total\_attacks}}} \times 100\% \tag{6}$$

where:

- D_{detected} is the number of replay attacks correctly identified and rejected by the validation mechanism
- $N_{\text{total\_attacks}}$ is the total number of replay attack attempts during the experiment

A higher Detection Rate, as defined in Equation 6, indicates stronger security performance. The value 100% represents perfect detection and 0% indicates that there are no attacks identified.

Previous research, such as that conducted by Al-Haija and Alsulami (2022), they measured the performance and effectiveness of methods for detecting replay attacks in RF vehicle access systems in terms of classification accuracy and Detection Rate by calculating the percentage of correctly identified malicious replayed messages.Similarly, Quintero et al. (2023) reported detection rates above 90% when applying SDR and machine learning models to replay attacks on smart cars. These studies confirm Detection Rate as a primary metric for assessing of replay attack defencing effectiveness.

3.14.2 Attack Success Rate (ASR)

The Attack Success Rate measures the proportion of replay attacks that bypass the validation mechanism and are incorrectly accepted as legitimate messages.

$$ASR = \frac{A_{\text{accepted}}}{N_{\text{total\_attacks}}} \times 100\% \tag{7}$$

where:

- A_{accepted} is the number of replay attack messages incorrectly accepted as legitimate by the validation mechanism.
- $N_{\text{total\_attacks}}$ is the total number of replay attack attempts during the experiment.

The Attack Success Rate, defined in Equation 7, is mathematically related to Detection Rate as:

$$ASR = 100\% - DR \tag{8}$$

A lower Attack Success Rate indicates more secure system. A value of 0% represents complete attack prevention, while 100% indicates a total system compromise.

Attack Success Rate is widely used to quantify the impact of replay attacks. Csikor et al. (2023) showed through their studies on RollBack attacks that even rolling-code RKE systems can be bypassed with high success rates and attack success the most meaningful security metric for these systems.Wouters et al. (2019) successfully compromised multiple commercial PKE systems in a real world vehicles including models like tesla and McLaren by demonstrating that an attack success is the most meaningful measure of system vulnerability. Similarly,Albinali and Azzedin (2023) reported that the replay attacks in RPL networks succeed in degrading packet delivery to below 60%. ASR therefore provides a direct measure of how successful an attacker is in defeating an validation mechanism

3.14.3 Latency

Latency measures the additional communication delay introduced by the security framework.

$$\text{Latency} = T_{secured} - T_{normal} \tag{9}$$

where:

- $T_{secured}$ is the message processing time with the validation mechanism enabled and
- T_{normal} is the message processing time without validation.

Latency is an important evaluation factor in lightweight IoT security since smart automobile systems rely on real-time communication with minimal latency. High overhead security methods could affect system performance and limit practical use in an resource constrained IoT environments. Kumar et al. (2024) identify latency and processing overhead as essential metrics when comparing lightweight authentication schemes for resource constrained IoT devices. Lazzaro et al. (2025) emphasize latency measurement when assessing authentication reliability in consumer IoT systems. These studies confirm that latency must be evaluated alongside any other security metrices.

3.14.4 False Rejection Rate (FRR)

The False Rejection Rate measures how often the system wrongly rejects a *genuine* message. This matters because a mechanism that blocks attacks but also frequently turns away the real owner is not usable in practice.

$$FRR = \frac{L_{\text{rejected}}}{N_{\text{total\_legit}}} \times 100\% \tag{10}$$

where:

- L_{rejected} is the number of genuine (legitimate) messages wrongly rejected by the validation mechanism, and
- $N_{\text{total\_legit}}$ is the total number of genuine messages sent.

A lower False Rejection Rate is better, with 0% meaning no genuine message is ever wrongly rejected. In this study a false rejection happens when two genuine messages pick the same nonce by chance, as explained in Section 3.7. The idea of reporting wrongful rejections alongside wrongful acceptances is taken from biometric systems, where the false-reject and false-accept rates are standard measures of how usable and how secure a system is (Jain et al., 2004).

4 Results and Analysis

This section presents the results of the experiments described in Section 3. All four validation methods were tested against five replay scenarios and three robustness scenarios (reset, desync and rollback). Every experiment was run as an isolated test with a fresh sender, receiver and attacker objects and each test was repeated 1000 times. Results are reported and recorded using the three metrics defined in Section 3.13 - Detection Rate (DR), Attack Success Rate (ASR), False Rejection Rate (FRR) and latency. Because the trials are repeated, each number is an average and is reported with a 95% confidence interval (CI), which shows how tightly the value is pinned down.

4.1 Results for Standard Replay Scenarios

For every method, the four standard attacks (delayed, multiple, out of order and counter-skip) produced exactly the same detection outcome, so they are summarised once per method in Table 6. so it is used only to report the false rejection rate on clean traffic and a baseline latency. The full per-scenario grid is included in the source-code output (`results_upgraded_grid.csv`).

Table 6: Results for the Standard Scenarios (1000 trials; DR/ASR identical across all four standard attacks)

Method	**DR (%)**	**ASR (%)**	**FRR (%)**	**Latency (μs)**
No Validation	0.0	100.0	0.00	$\approx$0.20
Nonce-Only	100.0	0.0	1.96 $\pm$0.27	$\approx$0.31
Counter-Only	100.0	0.0	0.00	$\approx$0.33
Hybrid	100.0	0.0	1.96 $\pm$0.27	$\approx$0.44

The results shows a clear difference between the unprotected baseline (no validation) and the three protected methods(nonce only, counter only and hybrid). The No Validation method accepted every replayed message, producing a 100% Attack Success Rate and a 0% Detection Rate. This confirms that without any freshness check, a smart car ECU is completely exposed to any replay attacks. All the three protected methods detected and rejected every replayed message and achieving a 100% Detection Rate and a 0% Attack Success Rate on all ffour standard scenarios.

The Nonce-Only and Hybrid methods wrongly rejected about 1.96% of *genuine* messages, because with a short 8-bit nonce two genuine messages sometimes pick the same value by chance and the second one is rejected as if it were a replay. The Counter-Only method has a 0% False Rejection Rate, because it uses no nonce and therefore cannot make this mistake. Figure 6 shows the Attack Success Rate for all four methods across all seven attack scenarios.

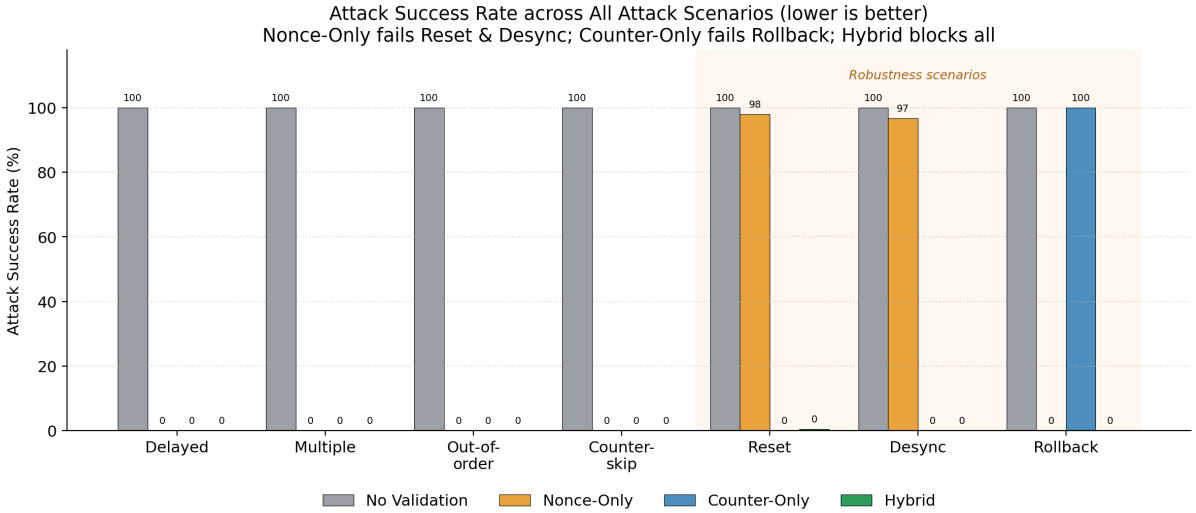

Figure 6: Attack Success Rate across all seven attack scenarios *(lower is better)*.

The four standard scenarios fail to differentiate between the three protected methods. A single lightweight freshness check is enough for ordinary replay attacks, where the attacker simply resends a message the ECU which already processed.

4.2 False Rejection Rate and the Nonce-Field Sweep

To show how the false rejection cost depends on the length of the nonce, the clean-traffic test was repeated while increasing the nonce field from 6 bits up to 16 bits. Table 7 reports the result.

Table 7: False Rejection Rate versus nonce field length (clean traffic, 1000 trials)

Nonce bits	**Possible values**	**Nonce-Only / Hybrid FRR (%)**	**Counter-Only FRR (%)**
6	64	6.88	0.0
8	256	1.96	0.0
10	1024	0.48	0.0
12	4096	0.12	0.0
14	16384	0.04	0.0
16	65536	0.01	0.0

The false rejection rate falls quickly as the nonce gets longer, because doubling the field roughly halves the chance of a collision. This means the false rejection cost is a design choice: a longer nonce almost removes it, at the price of a few more bits per message. The Counter-Only method stays at 0% throughout, because it has no nonce to collide. Figures 7 and 8 show this graphically.

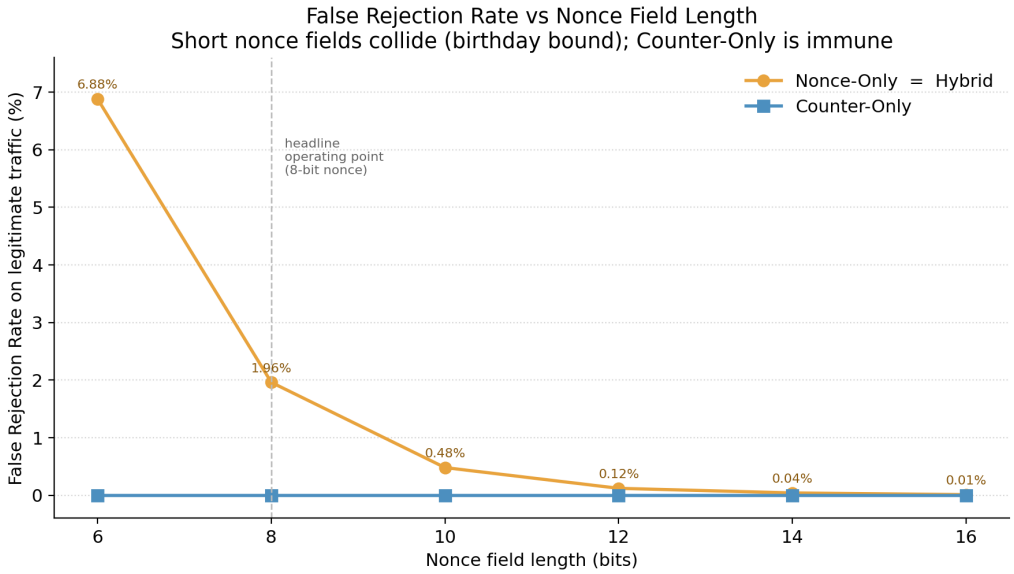

Figure 7: False Rejection Rate versus nonce field length. Longer nonce, fewer wrongly rejected genuine messages.

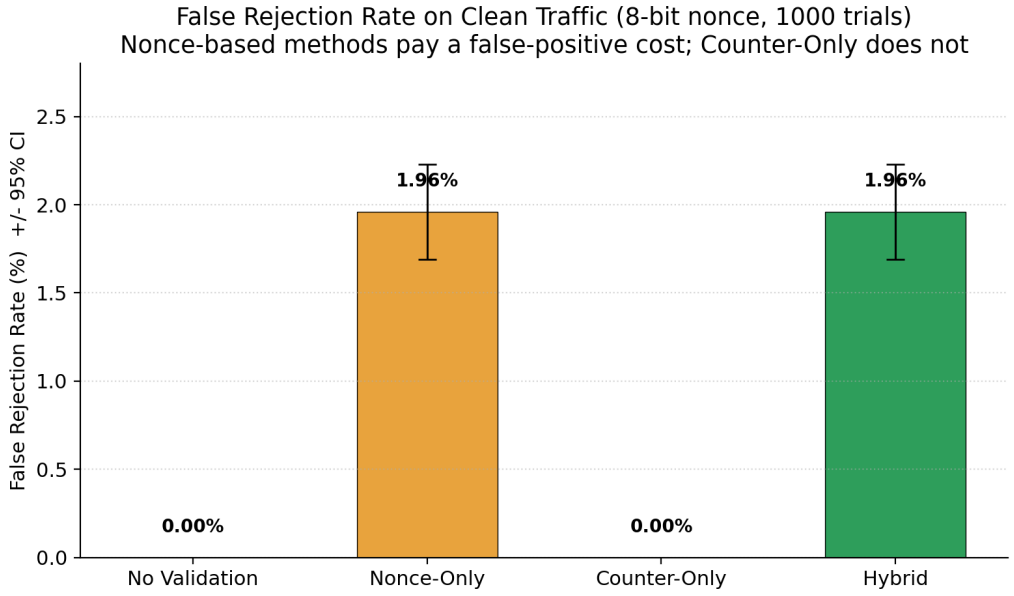

Figure 8: False Rejection Rate by method at the 8-bit headline setting. Counter-Only is structurally zero.

4.3 Results for Robustness Scenarios

The three robustness scenarios were designed to represent operating conditions that the standard tests often ignore.

1. An ECU power loss (reset).
2. A jamming attack (desync).
3. A counter-rollback attack (rollback).

The Table 8 reports the results.

Table 8: Results for the Reset, Desync and Rollback Robustness Scenarios (1000 trials per cell)

	Reset		**Desync**		**Rollback**	
Method	**DR (%)**	**ASR (%)**	**DR (%)**	**ASR (%)**	**DR (%)**	**ASR (%)**
No Validation	0.0	100.0	0.0	100.0	0.0	100.0
Nonce-Only	1.96	98.04	3.3	96.7	100.0	0.0
Counter-Only	100.0	0.0	100.0	0.0	0.0	100.0
Hybrid	99.61	0.39	100.0	0.0	100.0	0.0

Reset scenario. Nonce-Only failed almost completely, with a 98.04% Attack Success Rate. When ECU loses power the nonce list, which is stored in volatile RAM, is cleared, while the counter, which is stored in a non-volatile EEPROM, is preserved. After the reset the ECU has no record of any stored nonce, so almost every replayed message looks fresh and accepted. Counter-Only held fully (0% ASR) because the stored counter still rejects the old, lower counter values. Hybrid also held (0.39% ASR); the tiny non-zero value comes from a rare case where an earlier nonce collision had already nudged the stored counter, and it is reported honestly rather than rounded to zero.

Desync scenario. Nonce-Only again failed with a 96.7% Attack Success Rate. In this scenario the attacker jams one message in transit. That means, the attacker captures a message and the ECU never receives it, so its nonce is never stored. When the attacker later replays that jammed message, it appears fresh to the Nonce-Only and then accepted. Counter-Only and Hybrid both rejected the replayed message, because the stored counter has already moved past the jammed message's counter value.

Rollback scenario. This is the mirror image and it is the scenario that finally separates Counter-Only from Hybrid. The attack forces the stored counter backwards while leaving the nonce memory intact. Counter-Only then failed completely (100% ASR), because the old captured codes now look fresh again and are accepted. Nonce-Only held (0% ASR), because those replayed codes carry nonces it has already seen. Hybrid also held (0% ASR), because its nonce check catches the replays even though the counter was fooled.

Both robustness scenarios defeat Nonce-Only, but for different underlying reasons. In the reset scenario the ECU *had seen* the messages and then *forgot* them.

This is the key outcome of this project. The reset and desync scenarios defeat the nonce, but the counter survives them. The rollback scenario defeats the counter but the nonce survives it. No single freshness value is safe against all three faults, while the Hybrid which uses both survives them.

4.4 Security Capability Summary

Figure 9 summarises the security capability of each method as the number of attack scenarios it neutralised (kept Attack Success Rate at or below 1%) out of the seven attack

scenarios.

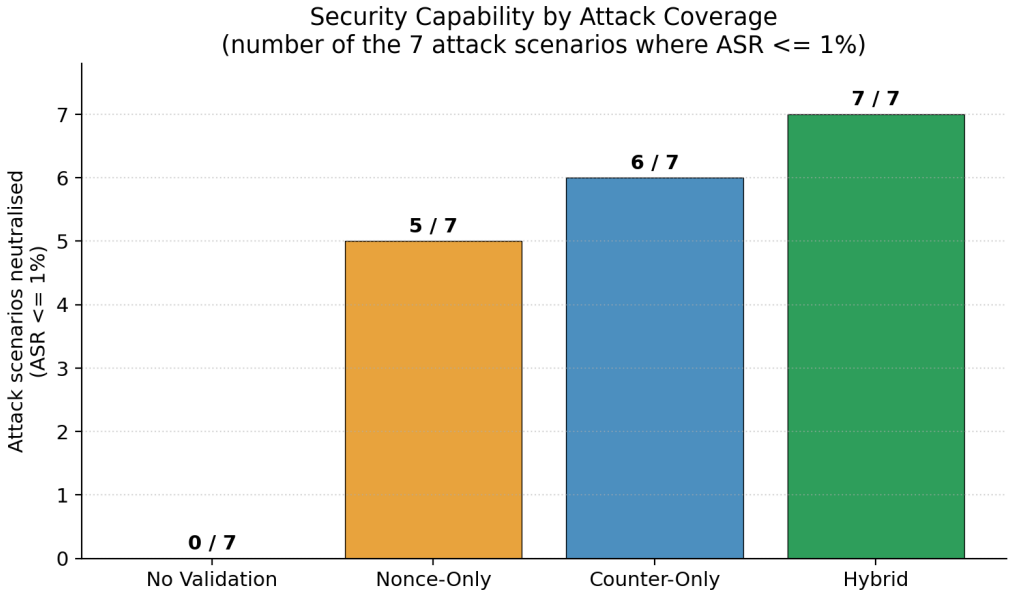

Figure 9: Security capability: number of the seven attack scenarios each method neutralised (Attack Success Rate $\leq 1\%$). Nonce-Only fails the *reset* and *desync* scenarios (5/7); Counter-Only fails only the *rollback* scenario (6/7); the Hybrid neutralises all seven.

No Validation defended zero scenarios. Nonce-Only defended five of the seven, but failed the reset and desync scenarios. Counter-Only neutralised six of seven,failing only rollback scenario. Hybrid neutralised all seven. In terms of pure detection capability on the scenarios tested, the Hybrid is now the strongest method on detection capability, because the rollback scenario exposes the one gap that the counter-only method has.

4.5 Latency Analysis

Latency was measured as the per-message validation time using Python's `time.perf_counter()`. Averaged across all scenarios, the No Validation baseline was the fastest at about 0.20 μs per message, as expected, since it performs no checks. The Nonce-Only and Counter-Only methods were close together at about 0.31-0.33 μs and the Hybrid was the slowest at about 0.44 μs because it performs both checks. Figure 10 shows the comparison.

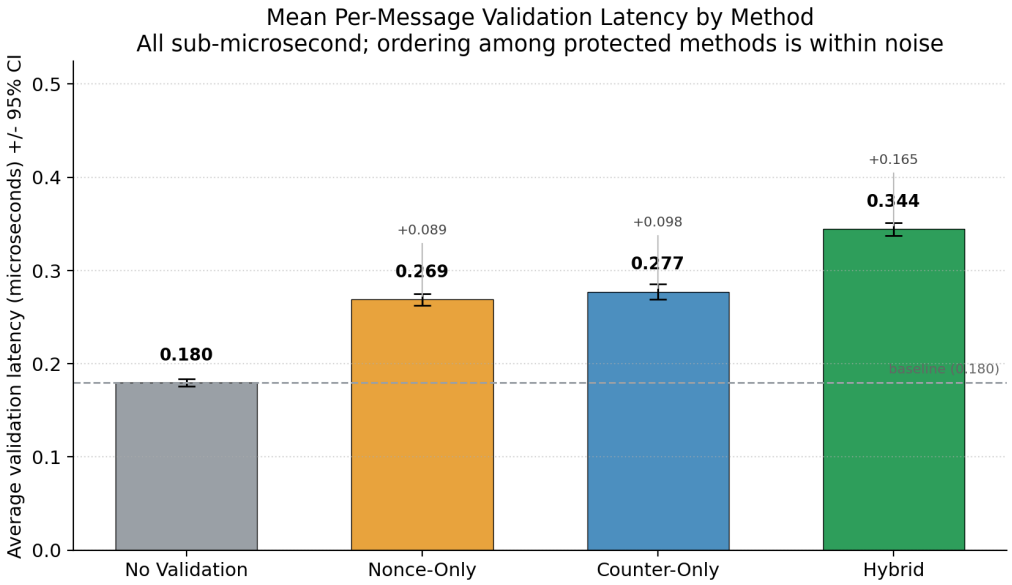

Figure 10: Mean per-message validation latency by method, averaged across all scenarios.

These values are extremely small, and at this sub-microsecond scale they vary slightly from run to run, so only two orderings are reliable: No Validation is always the fastest, and Hybrid is always the slowest. Even the largest gap, about 0.24 μs between the baseline and the Hybrid is tiny. By comparison, the physical action of a car door unlocking takes around 100 milliseconds, which is hundreds of thousands of times longer than any of these validation times. The latency cost of replay protection is therefore negligible in any practical scenario and security, not speed, is the deciding factor between these methods.

4.6 Hypothesis Evaluation

The four hypotheses stated in Section 1.5 can now be assessed against the results.

- **H1 (Detection Capability)** stated that the Hybrid would match or beat both single methods and strictly beat at least one. This is *supported*. The Hybrid neutralised 7 of 7 scenarios, against 5 of 7 for Nonce-Only and 6 of 7 for Counter-Only, so it's the strongest.

- **H2 (Attack Resistance)** stated that the Attack Success Rate would be lowest for the Hybrid compared with the baseline. This is *supported*. The Hybrid kept the Attack Success Rate at or near 0% in every scenario (worst case 0.39% on reset), against a 100% baseline.

- **H3 (Performance / Usability Trade-off)** predicted that the Hybrid would carry a measurable cost in both latency and false rejection. This is *supported*. The Hybrid had the highest latency of all methods and a 1.96% false rejection rate at the 8-bit setting. Both costs are real, confirming that the Hybrid is a security-usability trade-off and not a cost-free win.

- **H4 (Complementary Failure Modes)** predicted that the nonce-only method would fail on reset and desync, the counter-only method would fail on rollback and the Hybrid would handle all. This is *supported*. Nonce-Only failed reset and desync, Counter-Only failed rollback and the Hybrid held strong in all three cases.

4.7 Overall Comparison

The table 9 summarizes all metrics calculated in this article, allowing all to easily compare the four methodologies. The "security score" column represents the number of attack scenarios neutralized by each approach (attack success at or below 1%). This score is objective and prevents arbitrary weighting.

Table 9: Consolidated results for all four methods.

	Attack Success Rate (%)				**Security**	**FRR**	**Latency**
Method	**Std.**	**Reset**	**Desync**	**Rollback**	**score**	**(%)**	**(μs)**
No Validation	100	100	100	100	0/7	0.00	≈0.20
Nonce-Only	0	98.04	96.7	0	5/7	1.96	≈0.31
Counter-Only	0	0	0	100	6/7	0.00	≈0.33
Hybrid	0	0.39	0	0	**7/7**	1.96	≈0.44

lower is better for every column except the security score.

4.8 Summary of Findings

The experiments produced four main findings.

- First, any single lightweight freshness check the nonce or counter is enough to stop ordinary replay attacks, reducing the Attack Success Rate from 100% to 0% on all four standard scenarios.
- Second, the two single methods have mirror-image weak points under realistic issues: Nonce-Only fails on power loss and jamming, Counter-Only fails on rollback, and only the Hybrid withstands all three.
- Third, the Hybrid's protection has a measurable cost: a 1.96% false rejection rate at the 8-bit setting (which a longer nonce reduces to near zero) and the highest latency of all methods (still well under a microsecond and negligible against a 100 ms door action).
- Fourth, the results are now statistically grounded: every value is an average over 1000 paired trials with a 95% confidence interval and differences are tested only where chance could explain them.

It should also be stated plainly that this design does not stop every known attack. The RollJam attack, where an attacker jams and stores a genuine code and replays it later, defeats all four methods including the Hybrid, because the replayed code is genuinely fresh (a new nonce and a forward counter). Stopping RollJam requires a two-way challenge-response exchange rather than one-way freshness, which is identified as future work (Parameswarath & Sikdar, 2022, 2024).

5 Discussion

5.1 Interpretation of the Findings

The experiments provide direct responses to project questions. Using any of the lightweight freshness checks completely eliminates the possibility for replay, reducing replay attack success from 100% to 0% across all five common cases. When subjected to realistic fault conditions, the methods become separable in a *complementary* way. The nonce fails when its volatile memory is lost (reset) or not written (desync), whereas the counter fails when its state is reversed (rollback). Because these faults affect separate memory systems, no single attack can withstand all three types of fault conditions.Consequently, the hybrid design includes both methods to mitigating each of the scenarios since it is the only one that incorporates both types of methods. As a result, it is concluded that while the hybrid design does not necessarily possess a "stronger" attribute than any other generic technique, it does address a particular, observable gap that is present in both of the individual methods.

Each value given includes a 95% confidence interval from 1000 paired trials, serving as an error bar around the measurement. When two methods have detection or attack-success rates whose confidence intervals overlap, the difference is due to measurement noise and should not be interpreted as one being more secure than the other. In those circumstances, the approaches are differentiated not by the main security number, but by the other assessed costs like latency, memory and behavior under fault scenarios. This provides the foundation for comparing the protective approaches in this study. The typical attacks are statistically indistinguishable, the relevant variations arise in the false-rejection cost and which fault scenarios each approach survives.

5.2 Comparison with Previous Studies

The four most directly comparable studies in the literature each address one or two issues of the replay attack problem on RKE channels, but none reports the same trade-off this project measures. A summary is given in Table 10.

Table 10: Comparison with prior studies on replay protection for RKE channels

Study	**Approach**	**Scope**	**Not measured**
Al-Haija & Alsulami (2022)	Deep neural network detector on captured RF signals	Detection accuracy on a single replay variant	False rejection of genuine messages; computational cost
Quintero et al. (2023)	SDR + machine-learning classifier for replay detection	Detection rate above 90% on smart-car captures	Robustness under reset, desync or rollback; usability cost
Csikor et al. (2023)	RollBack attack analysis on rolling-code RKE	Demonstrates that high attack success is possible against modern RKE	Proposes no concrete lightweight defence on the same channel
Wouters et al. (2019)	Cryptanalysis of commercial PKE systems	Shows protocol-level breaks in deployed cars	One-way RKE channel; lightweight nonce/counter trade-off
This project	Controlled comparison of nonce, counter and hybrid validation on a constrained one-way RKE channel	Detection Rate, Attack Success Rate, False Rejection Rate and latency across 7 scenarios incl. reset, desync, rollback	RF physical-layer behaviour; adaptive attacker

5.3 Novelty and Contribution

This study's limited novelty has to be disclosed. Combining a random number with a serial number is not a novel idea. The 3GPP Authentication Vector uses the same design philosophy (use a serial number with a random challenge)(3GPP, 2020) as this conventional time varying parameter design(Menezes et al., 1996). There is no new cryptographic primitive created or identified by this study. An empirical analysis of the tradeoffs involved with an controlled, parallel, repeatable comparison of nonce, counter, hybrid and a no protection scenarios on a single limited, unidirectional RKE channel is the project's intended contribution. Additionally, this scientific study will reveal the attack success rate and the usually ignored usability costs (false rejection) for each nonce. Finally, this empirical measurement will determine how false rejection and the related usability costs of each nonce depend on the security parameters associated with the respective nonce and how it will do so under reset and desync and rollback conditions.

5.4 Strengths and Weakness

Strengths. The study's central strength is *defence in depth*. The hybrid model is the only configuration evaluated with no uncovered failure mode in threat model because the nonce and counter checks fail under different conditions and an attack must defeat both at once to succeed. This robustness is achieved at a microsecond measured latency.

The study provides a specific, evidence-based design recommendation by reporting false rejection with detection, addressing a usability cost that other detection-focused work often overlooks.

Weakness. The work has specific limits, specified simply. First, the hybrid does not stop a RollJam jam and store attack, because the replayed code has never reached the receiver, it is genuinely fresh to both checks and no receiver freshness mechanism can prevent it (Kamkar, 2015). Second, the study and experiment is a protocol level simulation, not a hardware testbed. The radio propagation, real jamming physics, clock drift, timing and power behaviour are out of scope so absolute latencies indicate relative cost only. Third, we consider that the attacker is non-adaptive . That means each scenario uses a fixed strategy, so the results are evidence under a stated threat model rather than a general security proof. Fourth, the 8 bit nonce is a deliberately pessimistic stress setting chosen to make the false-rejection cost visible. In a deployed system, they would use a wider field.

5.5 Impact on Legal, Cultural and Ethical Practices

Legal and regulatory: Vehicle cybersecurity is now subject to formal regulation. Standards such as ISO/SAE 21434, which defines cybersecurity engineering for road vehicles and UNECE Regulation No. 155, which makes a certified cybersecurity management system a condition of vehicle type approval, place a duty on manufacturers to demonstrate that foreseeable attacks (replay against keyless entry among them) have been considered and mitigated (International Organization for Standardization, 2021; United Nations Economic Commission for Europe, 2021). Work of this kind which evaluates freshness defences against named attack classes and reports the cost of each, supports exactly the evidence base such instruments expect. This is offered as context rather than legal advice, the precise applicability of these instruments varies by jurisdiction and would need to be confirmed for any specific deployment.

Cultural and usability: Remote keyless entry is a popular convenience looked on by consumers who cannot assess its security for themselves and keyless theft is a documented and growing problem whose financial cost falls on owners rather than on the manufacturers who chose the design. The false-rejection trade-off is a cultural as much as a technical issue. if a system that occasionally locks out a valid user creates trust and pushes that user toward insecure workarounds, such as repeatedly pressing the fob.

Ethical and data scope: The study reproduces only attack behaviour already available in the literature, introduces no new exploit capability or tools and was conducted entirely in simulation with no real vehicle, radio transmission or human participant involved. its purpose and its output are defensive. The use of synthetic data avoids collecting any personal or sensitive information and the deterministic, open, reproducible design is itself an ethical choice, allowing the claims to be verified rather than trusted. The full ethical scope of the work is set out in Section 3.12.

5.6 Implications

For low-power devices, designers should use a hybrid security system (nonce + counter) instead of a counter-only method, as single methods fail under realistic faults while the

hybrid option handles them all safely. When setting this up, the nonce field should be large enough to avoid accidentally blocking the legitimate messages and it must include an memory clearing policy so the device's storage wont overflow. Additionally, security tests need to be explicitly reporting these accidental blockages (false rejections) because silent failures create a massive hidden usability cost. And the main implications are, these improvements can be coded directly into existing devices right now without needing new hardware.

5.7 Summary

The results show that for ordinary replay attacks any single lightweight freshness check is sufficient. But the only the hybrid method survives the three realistic fault scenarios examined in this project. The cost of that broader coverage is small and quantifiable. A 1.96% False Rejection Rate at the 8-bit nonce setting and the highest validation latency among the four methods, which remains negligible against the physical door action time. Comparison with prior studies shows that the project's contribution is not a higher headline detection number, but the trade-off measurement on a constrained one-way RKE channel that earlier work did not report.

6 Conclusion

This project set out to address replay attacks in smart-car IoT authentication, which are serious problems because they bypass authentication without breaking the encryption. To address this, this study designed and evaluated a lightweight hybrid framework that combines nonce-based freshness with counter-based sequencing and compared it with against the nonce-only and counter-only methods and an unprotected baseline.

The hybrid framework has a small false-rejection rate of about 1.96% at the short 8-bit nonce setting, which is reduced to near zero with a longer nonce. It also has the highest validation latency of the methods tested, but that latency is still far too small to be significant against the time it takes to operate a real door lock. The honest conclusion is that the hybrid is a security-usability trade-off. It provides more protection at a measurable but acceptable cost.

The study also makes its limits clear. The hybrid does not prevent the RollJam attack, in which an attacker jams and stores a valid code before replaying it later because the replayed code is actually fresh on both the nonce and the counter. Defeating RollJam requires a twoway challenge-response exchange rather than one-way freshness.

Overall, this project contributes a working simulation and a defensible, evidence based design guideline for replay protection in a resource constrained smart car IoT systems, together with a clear account of where lightweight freshness helps and where it does not.

6.1 Recommendations

Based on the findings, the following recommendations are made. For vendors and manufacturers: adopt the hybrid nonce and counter check wherever a rollback or reset desync threat is credible.Increase the size of the nonce field to an explicit false-rejection budget rather than minimising it. Where a return channel can be added, a two-way challenge-response exchange should be layered on top to close the jam and store (RollJam) gap that oneway freshness cannot address. For researchers evaluating lightweight replay defences, report false rejection alongside detection and test under realistic fault conditions like power loss, jamming and counter rollback rather than clean replays alone, since these are the conditions under which single-mechanism defences actually separate.

6.2 Limitations and Future Work

Limitations

- **RollJam is not prevented**. In a RollJam attack scenario, the attacker jams and also stores genuine code and then replays it later (Kamkar, 2015). That code is genuinely fresh on both the nonce and the counter, so it defeats all four methods including the Hybrid. One-way freshness cannot solve this by design.
- **Simulation and protocol-level scope.** Because the threat model excludes RF-layer characteristics (signal strength, real jamming physics, propagation) and device-specific timing, absolute latencies indicate relative cost instead of special hardware.
- **Idealised attacker memory.** Perfect capture is considered, which is a deliberate worst case but not a realistic capture model.

Future work Methods to Defeat RollJam: The next step is to implement a two way exchange in which the ECU issues a fresh challenge the fob must answer, so a stored code cannot be replayed later. PUF-based mutual-authentication schemes already take this route and report resistance to replay, RollJam and even RollBack, at the cost of a return channel and extra hardwares (Parameswarath & Sikdar, 2022, 2024).

References

3GPP. (2020). *3G Security; Security Architecture (TS 33.102)* (tech. rep. No. TS 33.102) (Authentication vector combines a fresh sequence number (SQN) with a random challenge (RAND)). 3rd Generation Partnership Project (3GPP). https://www.3gpp.org/dynareport/33102.htm

Albinali, H., & Azzedin, F. (2023). Replay attacks in RPL-Based Internet of Things: Survey and Empirical Comparative Study. *Research Square*. https://doi.org/10.21203/rs.3.rs-3697695/v1

Al-Haija, Q. A., & Alsulami, A. A. (2022). Detection of Fake Replay Attack Signals on Remote Keyless Controlled Vehicles Using Pre-Trained Deep Neural Network. *Electronics*, *11*(20), 3376. https://doi.org/10.3390/electronics11203376

Alharthi, A. M., & Altuwaijri, F. S. (2025). Lightweight IoT authentication protocol using PUFS in smart manufacturing industry. *Electronics*, *14*(9), 1788. https://doi.org/10.3390/electronics14091788

Arcuri, A., & Briand, L. (2014). A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability*, *24*(3), 219–250. https://doi.org/10.1002/stvr.1486

Bianchi, T., Brighente, A., Conti, M., & Pavan, E. (2025). SOK: Stealing cars since Remote Keyless Entry introduction and how to defend from it. *ArXiv.org*. https://doi.org/10.48550/arxiv.2505.02713

Byeon, H. (2026). A security analysis and lightweight hardening of the authentication chains protocol for the internet of things. *International Journal of Safety and Security Engineering*, *16*(1). https://doi.org/10.18280/ijsse.160114

Coston, I., Plotnizky, E., & Nojoumian, M. (2025). Comprehensive Study of IoT Vulnerabilities and Countermeasures. *Applied Sciences*, *15*(6), 3036. https://doi.org/10.3390/app15063036

Csikor, L., Lim, H. W., Wong, J. W., Ramesh, S., Parameswarath, R. P., & Chan, M. C. (2023). RollBack: a new Time-Agnostic replay attack against the automotive remote keyless entry systems. *ACM Transactions on Cyber-Physical Systems*, *8*(1), 1–25. https://doi.org/10.1145/3627827

Djinko, I. A. R., Kacem, T., & Girma, A. (2022). Blockchain-based Approach to thwart Replay Attacks targeting Remote Keyless Entry Systems. *2022 International Conference on Engineering and Emerging Technologies (ICEET)*, 1–6. https://doi.org/10.1109/iceet56468.2022.10007335

Dworkin, M. J. (2007). *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC* (tech. rep. No. NIST Special Publication 800-38D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-38D

Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., & Shalmani, M. T. M. (2008). *Physical Cryptanalysis of KeeLoq Code Hopping Applications* (tech. rep. No. Report 2008/058). IACR Cryptology ePrint Archive. https://eprint.iacr.org/2008/058.pdf

El-Hajj, M., Fadlallah, A., Chamoun, M., & Serhrouchni, A. (2019). A Survey of Internet of Things (IoT) Authentication Schemes. *Sensors*, *19*(5), 1141. https://doi.org/10.3390/s19051141

International Organization for Standardization. (2021). *Iso/sae 21434:2021 road vehicles — cybersecurity engineering*. ISO/SAE International.

Jain, A. K., Ross, A., & Prabhakar, S. (2004). An Introduction to Biometric Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, *14*(1), 4–20. https://doi.org/10.1109/TCSVT.2003.818349

Kamkar, S. (2015). Drive It Like You Hacked It: New Attacks and Tools to Wirelessly Steal Cars. https://samy.pl/defcon2015/

Køien, G. M. (2015). A Brief Survey of Nonces and Nonce Usage. *SECURWARE 2015: The Ninth International Conference on Emerging Security Information, Systems and Technologies*, 85–91.

Krentz, K.-F., & Voigt, T. (2024). More lightweight, yet stronger: Revisiting OSCORE's replay protection. https://doi.org/10.14722/sdiotsec.2024.23003

Kumar, S., Kumar, D., Dangi, R., Choudhary, G., Dragoni, N., & You, I. (2024). A review of lightweight security and privacy for Resource-Constrained IoT devices. *Computers, materials & continua/Computers, materials & continua (Print)*, *78*(1), 31–63. https://doi.org/10.32604/cmc.2023.047084

Lazzaro, S., De Angelis, V., Mandalari, A. M., & Buccafurri, F. (2024). Is your kettle smarter than a hacker? A scalable tool for assessing replay attack vulnerabilities on consumer IoT devices. *EEE International Conference on Pervasive Computing and Communications (PerCom)*, 114–124. https://doi.org/10.1109/percom59722.2024.10494466

Lazzaro, S., De Angelis, V., Mandalari, A. M., & Buccafurri, F. (2025). A black-box assessment of authentication and reliability in consumer IoT devices. *Pervasive and Mobile Computing*, *110*, 102045. https://doi.org/10.1016/j.pmcj.2025.102045

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography* [Section 10.3.1, time-variant parameters (random numbers, sequence numbers, timestamps)]. CRC Press. https://cacr.uwaterloo.ca/hac/

Microchip Technology Inc. (2011). *HCS300 KeeLoq Code Hopping Encoder* (tech. rep. No. Datasheet DS21137G) (Documents the 16-code single-operation window, 32K resynchronisation window, and 16-bit synchronisation counter (Fig. 7-3)). Microchip Technology Inc. http://ww1.microchip.com/downloads/en/devicedoc/21137f.pdf

Parameswarath, R. P., & Sikdar, B. (2022). An Authentication Mechanism for Remote Keyless Entry Systems in Cars to Prevent Replay and RollJam Attacks. *2022 IEEE Intelligent Vehicles Symposium (IV)*, 1725–1730. https://doi.org/10.1109/IV51971.2022.9827256

Parameswarath, R. P., & Sikdar, B. (2024). A Secure PUF-Based Authentication Protocol for Remote Keyless Entry Systems in Cars [First protocol to address the RollBack attack]. *IEEE Transactions on Vehicular Technology*. https://ieeexplore.ieee.org/document/10439658

Quintero, J. C. M., Cuesta, E. P. E., & Lopez, L. J. R. (2023). A new method for the detection and identification of the replay attack on cars using SDR technology and classification algorithms. *Results in Engineering*, *19*, 101243. https://doi.org/10.1016/j.rineng.2023.101243

Rakshit, H., Bhandari, R., & Banerjee, S. (2025). Lightweight Session-Key rekeying framework for secure IoT-Edge communication. *ArXiv.org*. http://arxiv.org/abs/2511.02924

United Nations Economic Commission for Europe. (2021). Un regulation no. 155 — uniform provisions concerning the approval of vehicles with regard to cyber security and cyber security management system.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer. https://doi.org/10.1007/978-3-642-29044-2

Wouters, L., Marin, E., Ashur, T., Gierlichs, B., & Preneel, B. (2019). Fast, furious and insecure: passive keyless entry and start systems in modern supercars. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 66–85. https://doi.org/10.46586/tches.v2019.i3.66-85

Zenner, E. (2009). Nonce Generators and the Nonce Reset Problem. *Information Security (ISC 2009)*, *5735*, 411–426. https://doi.org/10.1007/978-3-642-04474-8_33

Zhang, X., & Tsou, T. (2012, January). *IPsec Anti-Replay Algorithm without Bit Shifting* (RFC No. 6479). Internet Engineering Task Force (IETF). RFC Editor. https://doi.org/10.17487/RFC6479

A Appendix A: Source Code

The complete source code for the simulation is provided below. All modules are written in Python 3 using only the standard library. The code is also maintained in a public Git repository for transparency and reproducibility.

message_final.py

Message structure (unchanged from v1).

```
"""
message_final.py  (v2)
Defines the Message structure exchanged between the key fob and the car ECU.
This module is UNCHANGED from v1 - the message fields were already sufficient.
"""

from dataclasses import dataclass
from datetime import datetime


@dataclass
class Message:
    """A command message transmitted from the key fob to the car ECU."""
    command: str       # UNLOCK / LOCK / START
    nonce: int         # freshness token (random per message)
    counter: int       # monotonic sequence number
    timestamp: float   # creation time (seconds)

    def __str__(self) -> str:
        return f"[{self.command}] Counter={self.counter} Nonce={self.nonce}"


def create_message(command: str, nonce: int, counter: int) -> Message:
    """Construct a Message stamped with the current time."""
    return Message(
        command=command,
        nonce=nonce,
        counter=counter,
        timestamp=datetime.now().timestamp(),
    )
```

Listing 1: message_final.py

sender_final.py

Key fob (sender). v2 adds a configurable nonce width and a seedable generator.

```
"""
sender_final.py  (v2)
Simulates the smart-car key fob.

WHAT CHANGED IN v2
The nonce field is now CONFIGURABLE (nonce_bits). v1 always drew a 6-digit
nonce from a space so large that two legitimate messages never collided, which
is why every result was a perfect 100% / 0%. Real lightweight RF devices use
SHORT nonce fields, and short fields collide (birthday bound). A configurable
field lets us measure the False Rejection Rate as a function of nonce length.
An optional rng (random.Random) can be injected so each trial uses its own
seeded generator, making trials reproducible AND paired across methods.
"""

import random
from message_final import Message, create_message


class KeyFob:
    """Key fob that generates command messages for transmission to the ECU."""

    def __init__(self, nonce_bits: int = 24, rng: random.Random = None):
        """
        nonce_bits : width of the nonce field; nonce drawn from [0, 2^bits - 1].
                     Small values (6-10) cause realistic birthday collisions.
        rng        : optional seeded random.Random for reproducible/paired trials.
        """
        self.counter = 0
        self.nonce_bits = nonce_bits
        self.nonce_max = (1 << nonce_bits) - 1
        self._rng = rng if rng is not None else random.Random()

    def _generate_nonce(self) -> int:
        return self._rng.randint(0, self.nonce_max)

    def _send_command(self, command: str) -> Message:
        self.counter += 1
        return create_message(
            command=command,
            nonce=self._generate_nonce(),
            counter=self.counter,
        )

    def unlock(self) -> Message:
```

```
45          return self._send_command("UNLOCK")
46
47      def lock(self) -> Message:
48          return self._send_command("LOCK")
49
50      def start(self) -> Message:
51          return self._send_command("START")
52
53      def reset(self) -> None:
54          """Reset the counter to zero (simulates key-fob battery removal
    )."""
55          self.counter = 0
```

Listing 2: sender_final.py

receiver_final.py

Car ECU (receiver). v2 adds the counter window and the rollback event.

```
"""
receiver_final.py  (v2)
Simulates the car ECU and its four validation methods.

WHAT CHANGED IN v2
1. COUNTER ACCEPTANCE WINDOW. v1 accepted any strictly-increasing counter.
   Real rolling-code RKE accepts counters only within a finite forward window:
       last_counter < C <= last_counter + counter_window
2. rollback() METHOD. Models the RollBack attack: the counter is forced backwards while the nonce memory survives - the mirror of volatile_reset().
The False Rejection behaviour is NOT injected here; it emerges from the unchanged nonce check when the sender uses a short, collision-prone field.
"""

from typing import Set
from message_final import Message


class CarECU:
    """Car ECU that validates incoming messages using a configurable method."""

    METHOD_NO_VALIDATION = 1
    METHOD_NONCE_ONLY = 2
    METHOD_COUNTER_ONLY = 3
    METHOD_HYBRID = 4

    def __init__(self, method: int = METHOD_NO_VALIDATION, counter_window: int = 256):
        self.method = method
        self.counter_window = counter_window      # forward window size W
        self.nonce_list: Set[int] = set()
        self.last_counter: int = 0
        self.accepted_count: int = 0
        self.rejected_count: int = 0

    def receive(self, msg: Message) -> bool:
        if self.method == self.METHOD_NO_VALIDATION:
            result = self._no_validation(msg)
        elif self.method == self.METHOD_NONCE_ONLY:
            result = self._nonce_only(msg)
        elif self.method == self.METHOD_COUNTER_ONLY:
            result = self._counter_only(msg)
        elif self.method == self.METHOD_HYBRID:
            result = self._hybrid(msg)
        else:
            raise ValueError(f"Unknown validation method: {self.method}")
        if result:
            self.accepted_count += 1
```

```
48          else:
49              self.rejected_count += 1
50          return result
51
52      def _no_validation(self, msg: Message) -> bool:
53          """Accept everything. Baseline control case."""
54          return True
55
56      def _nonce_only(self, msg: Message) -> bool:
57          """Reject if the nonce was seen before, else accept and store
    it."""
58          if msg.nonce in self.nonce_list:
59              return False
60          self.nonce_list.add(msg.nonce)
61          return True
62
63      def _counter_only(self, msg: Message) -> bool:
64          """Accept only if the counter is inside the forward window."""
65          if self.last_counter < msg.counter <= self.last_counter + self.
    counter_window:
66              self.last_counter = msg.counter
67              return True
68          return False
69
70      def _hybrid(self, msg: Message) -> bool:
71          """Two sequential checks: nonce freshness AND counter window.""
    "
72          if msg.nonce in self.nonce_list:                    # check 1
73              return False
74          if not (self.last_counter < msg.counter <= self.last_counter +
    self.counter_window):
75              return False                                    # check 2
76          self.nonce_list.add(msg.nonce)
77          self.last_counter = msg.counter
78          return True
79
80      def reset(self) -> None:
81          """Full reset of all state. Used between experiments for
    isolation."""
82          self.nonce_list.clear()
83          self.last_counter = 0
84          self.accepted_count = 0
85          self.rejected_count = 0
86
87      def volatile_reset(self) -> None:
88          """
89          Realistic ECU power loss (reset scenario).
90          The nonce list (volatile RAM) is cleared; the last counter
91          (non-volatile EEPROM) is preserved. Defeats Nonce-Only.
92          """
93          self.nonce_list.clear()
94          # last_counter intentionally preserved
95
96      def rollback(self) -> None:
97          """
98          RollBack attack (rollback scenario).
99          The counter is forced backwards to its initial value while the
    nonce
```

```
100         memory survives. Defeats Counter-Only; Nonce-Only and Hybrid
    hold.
101         """
102         self.last_counter = 0
103         # nonce_list intentionally preserved
```

Listing 3: receiver_final.py

attacker_final.py

Attacker and replay strategies. v2 adds replay_all() for reset/rollback.

```
"""
attacker_final.py  (v2)
Simulates an adversary positioned between the key fob and the ECU.
The attacker is deliberately simple (worst-case passive capture-and-
    replay).
silent_capture() models a jamming attacker that records a message
    without
letting it reach the ECU (used by the desync scenario). replay_all() is
     used
by the reset and rollback scenarios.
"""

from typing import List
from message_final import Message


class Attacker:
    """Adversary that captures and replays messages between key fob and
    ECU."""

    def __init__(self):
        self.captured_messages: List[Message] = []

    def intercept(self, msg: Message) -> Message:
        """Capture a message in transit and forward it unchanged to the
    ECU."""
        self.captured_messages.append(msg)
        return msg

    def silent_capture(self, msg: Message) -> None:
        """Capture a message WITHOUT forwarding it (jamming)."""
        self.captured_messages.append(msg)

    def delayed_replay(self, index: int = 0) -> List[Message]:
        """Replay a single captured message at the given index."""
        if index >= len(self.captured_messages):
            return []
        return [self.captured_messages[index]]

    def multiple_replay(self, index: int = 0, count: int = 5) -> List[
    Message]:
        """Replay the same captured message multiple times in
    succession."""
        if index >= len(self.captured_messages):
            return []
        return [self.captured_messages[index]] * count

    def out_of_order_replay(self) -> List[Message]:
        """Replay all captured messages in reverse chronological order.
    """
        return list(reversed(self.captured_messages))

    def counter_skip_replay(self) -> List[Message]:
        """Replay the oldest captured message (lowest counter)."""
        if not self.captured_messages:
```

```
48              return []
49          return [self.captured_messages[0]]
50
51      def replay_all(self) -> List[Message]:
52          """Replay every captured message (used by reset and rollback
    scenarios)."""
53          return list(self.captured_messages)
54
55      def clear_captured(self) -> None:
56          self.captured_messages.clear()
```

Listing 4: attacker_final.py

main_final.py

Experiment controller (v2). The configuration, the per-trial simulation, the confidence-interval helper and the paired-trial driver are shown. The interactive menu and the CSV-writing helpers are omitted here for brevity; the full version is in the Git repository.

```
"""
main_final.py  (v2)
Experiment controller for the upgraded replay-attack simulation.
- 1000 paired, seeded trials per (method, scenario) cell; mean +/- 95% CI.
- Four metrics: Detection Rate, Attack Success Rate, False Rejection Rate, latency.
- Standard library only (statistics, math) - no numpy, no scipy.
- Writes a CSV per experiment (grid, per-method, per-scenario, sweep, stats).
"""

import os, csv, sys, math, time, random, statistics
from sender_final import KeyFob
from receiver_final import CarECU
from attacker_final import Attacker

# Always write CSVs next to THIS script, not the terminal's current folder.
HERE = os.path.dirname(os.path.abspath(__file__))

# ---- configuration ----
NUM_LEGIT = 10                  # legitimate messages per trial
MULTIPLE_REPLAY_COUNT = 5
NUM_TRIALS = 1000
BASE_SEED = 42
HEADLINE_NONCE_BITS = 8         # short, constrained-device nonce field
COUNTER_WINDOW = 256            # forward acceptance window W
SWEEP_BITS = [6, 8, 10, 12, 14, 16]
Z95 = 1.96                      # 95% CI multiplier (large-sample normal approx.)

METHODS = [
    (CarECU.METHOD_NO_VALIDATION, "No Validation"),
    (CarECU.METHOD_NONCE_ONLY, "Nonce-Only"),
    (CarECU.METHOD_COUNTER_ONLY, "Counter-Only"),
    (CarECU.METHOD_HYBRID, "Hybrid"),
]

SCENARIOS = [
    "no_attack", "delayed_replay", "multiple_replay", "out_of_order",
    "counter_skip", "reset_attack", "desync_attack", "counter_rollback",
]


def run_trial(method_id, scenario, stream, counter_window):
    """Run one (method, scenario) trial on a pre-built message stream."""
    car = CarECU(method=method_id, counter_window=counter_window)
    attacker = Attacker()
    legit_sent = legit_accepted = 0
```

```
    attacks_total = attacks_accepted = 0
    latencies = []

    def deliver(msg):
        nonlocal legit_sent, legit_accepted
        attacker.intercept(msg)                  # adversary records on the wire
        t0 = time.perf_counter()
        ok = car.receive(msg)
        latencies.append((time.perf_counter() - t0) * 1_000_000)
        legit_sent += 1
        legit_accepted += int(ok)

    if scenario in ("no_attack", "delayed_replay", "multiple_replay",
                    "out_of_order", "counter_skip"):
        for m in stream:
            deliver(m)
        if scenario == "no_attack":
            attack_msgs = []
        elif scenario == "delayed_replay":
            attack_msgs = attacker.delayed_replay(0)
        elif scenario == "multiple_replay":
            attack_msgs = attacker.multiple_replay(0, MULTIPLE_REPLAY_COUNT)
        elif scenario == "out_of_order":
            attack_msgs = attacker.out_of_order_replay()
        else:  # counter_skip
            attack_msgs = attacker.counter_skip_replay()

    elif scenario == "reset_attack":
        for m in stream:
            deliver(m)
        car.volatile_reset()                     # wipe nonce RAM, keep counter
        attack_msgs = attacker.replay_all()

    elif scenario == "counter_rollback":
        for m in stream:
            deliver(m)
        car.rollback()                           # force counter back, keep nonces
        attack_msgs = attacker.replay_all()

    elif scenario == "desync_attack":
        half = len(stream) // 2
        for m in stream[:half]:
            deliver(m)
        jammed = stream[half]
        attacker.silent_capture(jammed)          # captured, never delivered
        for m in stream[half + 1:]:
            deliver(m)
        attack_msgs = [jammed]
    else:
        raise ValueError(scenario)

    for m in attack_msgs:
        t0 = time.perf_counter()
```

```
99          ok = car.receive(m)
100         latencies.append((time.perf_counter() - t0) * 1_000_000)
101         attacks_total += 1
102         attacks_accepted += int(ok)
103
104     return {
105         "legit_sent": legit_sent,
106         "legit_accepted": legit_accepted,
107         "attacks_total": attacks_total,
108         "attacks_accepted": attacks_accepted,
109         "mean_latency": statistics.mean(latencies) if latencies else
    0.0,
110     }
111
112
113 def ci95(values):
114     """95% CI half-width (large-sample normal approx; ~ t at n=1000).""
    "
115     n = len(values)
116     if n < 2:
117         return 0.0
118     sd = statistics.stdev(values)
119     return Z95 * sd / math.sqrt(n)
120
121
122 def run_cell(method_id, scenario, nonce_bits, counter_window,
    num_trials):
123     """Run num_trials paired trials and aggregate metrics with CIs."""
124     frr, dr, asr, lat = [], [], [], []
125     for t in range(num_trials):
126         rng = random.Random(BASE_SEED + t)           # paired across
    methods
127         kf = KeyFob(nonce_bits=nonce_bits, rng=rng)
128         stream = [kf.unlock() for _ in range(NUM_LEGIT)]
129         r = run_trial(method_id, scenario, stream, counter_window)
130         if r["legit_sent"]:
131             frr.append((1 - r["legit_accepted"] / r["legit_sent"]) *
    100)
132         if r["attacks_total"]:
133             d = (1 - r["attacks_accepted"] / r["attacks_total"]) * 100
134             dr.append(d)
135             asr.append(100 - d)
136         lat.append(r["mean_latency"])
137     mean = lambda xs: statistics.mean(xs) if xs else None
138     return {
139         "frr_mean": mean(frr), "frr_ci": ci95(frr) if frr else None,
140         "dr_mean": mean(dr),
141         "asr_mean": mean(asr), "asr_ci": ci95(asr) if asr else None,
142         "lat_mean": mean(lat), "lat_ci": ci95(lat),
143     }
144
145
146 def paired_t(a, b):
147     """Manual paired t-statistic (no scipy). Returns (t, df, mean_diff)
    ."""
148     diffs = [x - y for x, y in zip(a, b)]
149     n = len(diffs)
150     md = statistics.mean(diffs)
```

```
    sd = statistics.stdev(diffs) if n > 1 else 0.0
    se = sd / math.sqrt(n) if sd else 0.0
    t = md / se if se else float("inf")
    return t, n - 1, md


# The interactive menu (options 1-9) and the CSV-writing helpers that save
# results_upgraded_grid.csv, the per-method files, the per-scenario files,
# results_nonce_sweep.csv and results_statistics.csv are omitted here for
# brevity. See the Git repository for the complete file.

if __name__ == "__main__":
    random.seed(BASE_SEED)
    # run_grid(); run_sweep(); run_stats(); ...  (see repository)
```

Listing 5: main_final.py (core)

Plot_Graph.py

Experiment controller.

```
"""
plot_results_final.py  (v2)
===========================
Generates the upgraded result figures from the v2 simulation CSVs.

Input : results_upgraded_grid.csv, results_nonce_sweep.csv  (from main_final.py)
Output: five PNGs ->
    fig1_asr_all_v2.png        ASR across all 7 attack scenarios (incl. rollback)
    fig2_frr_vs_nonce.png      FRR vs nonce field length  (the new "money" figure)
    fig3_frr_by_method.png     FRR on clean traffic, with 95% CI (false positives)
    fig4_latency_v2.png        Mean validation latency + 95% CI + overhead
    fig5_capability_v2.png     Attack scenarios neutralised (ASR <= 1%)

Requirements: matplotlib, numpy
"""

import os
import csv
import sys
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

# Always read CSVs from and write PNGs to THIS script's folder, regardless of
# the terminal's current working directory.
HERE = os.path.dirname(os.path.abspath(__file__))

COLOURS = {
    "No Validation": "#9aa0a6",
    "Nonce-Only":    "#e8a33d",
    "Counter-Only":  "#4c8fbf",
    "Hybrid":        "#2e9e5b",
}
ORDER = ["No Validation", "Nonce-Only", "Counter-Only", "Hybrid"]
plt.rcParams.update({"font.family": "DejaVu Sans", "font.size": 11})


def f(x):
    return None if x in ("", None) else float(x)


def load(name):
    path = os.path.join(HERE, name)
    if not os.path.exists(path):
        print(f"ERROR: cannot find {name} in:\n  {HERE}\n")
        print("Generate the CSVs first by running the simulation, e.g.:")
        print("  python main_final.py --all")
```

```
        print("(or run the menu and choose [8] Run Everything, "
              "or [1] then [6]).")
        sys.exit(1)
    with open(path) as fh:
        return list(csv.DictReader(fh))


grid = load("results_upgraded_grid.csv")
sweep = load("results_nonce_sweep.csv")

asr = {(r["method"], r["scenario"]): f(r["asr_mean"]) for r in grid}
asr_ci = {(r["method"], r["scenario"]): f(r["asr_ci"]) for r in grid}
frr = {(r["method"], r["scenario"]): f(r["frr_mean"]) for r in grid}
frr_ci = {(r["method"], r["scenario"]): f(r["frr_ci"]) for r in grid}
lat = {(r["method"], r["scenario"]): f(r["lat_mean"]) for r in grid}
lat_ci = {(r["method"], r["scenario"]): f(r["lat_ci"]) for r in grid}


# ======================================================================
# FIGURE 1 : ASR across all 7 attack scenarios
# ======================================================================
scen = ["delayed_replay", "multiple_replay", "out_of_order", "counter_skip",
        "reset_attack", "desync_attack", "counter_rollback"]
labels = ["Delayed", "Multiple", "Out-of-\norder", "Counter-\nskip",
          "Reset", "Desync", "Rollback"]

fig, ax = plt.subplots(figsize=(12, 5.4))
x = np.arange(len(scen))
bw = 0.2
for i, m in enumerate(ORDER):
    vals = [asr[(m, s)] for s in scen]
    bars = ax.bar(x + (i - 1.5) * bw, vals, bw, label=m, color=COLOURS[m],
                  edgecolor="black", linewidth=0.4)
    for b, v in zip(bars, vals):
        ax.text(b.get_x() + b.get_width() / 2, v + 1.5, f"{v:.0f}",
                ha="center", va="bottom", fontsize=7)
ax.axvspan(3.5, 6.5, color="#fff4e6", alpha=0.6, zorder=0)
ax.text(5.0, 109, "Robustness scenarios", ha="center", fontsize=9,
        style="italic", color="#a8631b")
ax.set_xticks(x); ax.set_xticklabels(labels)
ax.set_ylabel("Attack Success Rate (%)"); ax.set_ylim(0, 118)
ax.set_title("Attack Success Rate across All Attack Scenarios (lower is better)\n"
             "Nonce-Only fails Reset & Desync; Counter-Only fails Rollback; "
             "Hybrid blocks all")
ax.legend(ncol=4, loc="upper center", frameon=False, bbox_to_anchor=(0.5, -0.13))
for s in ("top", "right"):
    ax.spines[s].set_visible(False)
ax.grid(axis="y", linestyle=":", alpha=0.5)
fig.tight_layout(); fig.savefig(os.path.join(HERE, "fig1_asr_all_v2.png"), dpi=200, bbox_inches="tight")
plt.close(fig)


```

```
# ======================================================================
# FIGURE 2 : FRR vs nonce field length  (money figure)
# ======================================================================
bits = [int(r["nonce_bits"]) for r in sweep]
n_frr = [float(r["nonce_only_frr"]) for r in sweep]
c_frr = [float(r["counter_only_frr"]) for r in sweep]

fig, ax = plt.subplots(figsize=(9, 5.2))
ax.plot(bits, n_frr, "-o", color=COLOURS["Nonce-Only"], linewidth=2,
        markersize=7, label="Nonce-Only  =  Hybrid")
ax.plot(bits, c_frr, "-s", color=COLOURS["Counter-Only"], linewidth=2,
        markersize=7, label="Counter-Only")
for bx, by in zip(bits, n_frr):
    ax.text(bx, by + 0.18, f"{by:.2f}%", ha="center", fontsize=8,
            color="#8a5a10")
# highlight the headline 8-bit operating point
ax.axvline(8, color="#bbb", linestyle="--", linewidth=1)
ax.text(8.15, 6.2, "headline\noperating point\n(8-bit nonce)", fontsize
    =8,
        color="#666", va="top")
ax.set_xlabel("Nonce field length (bits)")
ax.set_ylabel("False Rejection Rate on legitimate traffic (%)")
ax.set_title("False Rejection Rate vs Nonce Field Length\n"
             "Short nonce fields collide (birthday bound); Counter-Only
    is immune")
ax.set_xticks(bits); ax.set_ylim(-0.4, 7.6)
ax.legend(frameon=False, loc="upper right")
for s in ("top", "right"):
    ax.spines[s].set_visible(False)
ax.grid(axis="y", linestyle=":", alpha=0.5)
fig.tight_layout(); fig.savefig(os.path.join(HERE, "fig2_frr_vs_nonce.
    png"), dpi=200, bbox_inches="tight")
plt.close(fig)


# ======================================================================
# FIGURE 3 : FRR by method on clean traffic (with 95% CI)
# ======================================================================
fig, ax = plt.subplots(figsize=(8, 4.8))
vals = [frr[(m, "no_attack")] for m in ORDER]
errs = [frr_ci[(m, "no_attack")] for m in ORDER]
bars = ax.bar(ORDER, vals, color=[COLOURS[m] for m in ORDER],
              edgecolor="black", linewidth=0.5, width=0.6,
              yerr=errs, capsize=5, error_kw={"elinewidth": 1.1})
for b, v in zip(bars, vals):
    ax.text(b.get_x() + b.get_width() / 2, v + 0.12, f"{v:.2f}%",
            ha="center", va="bottom", fontsize=10, fontweight="bold")
ax.set_ylabel("False Rejection Rate (%)  +/- 95% CI")
ax.set_ylim(0, 2.8)
ax.set_title("False Rejection Rate on Clean Traffic (8-bit nonce, 1000
    trials)\n"
             "Nonce-based methods pay a false-positive cost; Counter-
    Only does not")
for s in ("top", "right"):
    ax.spines[s].set_visible(False)
ax.grid(axis="y", linestyle=":", alpha=0.5)
fig.tight_layout(); fig.savefig(os.path.join(HERE, "fig3_frr_by_method.
    png"), dpi=200, bbox_inches="tight")
```

```
plt.close(fig)


# ============================================================================
# FIGURE 4 : mean validation latency + 95% CI + overhead
# ============================================================================
# average each method's latency across all scenarios
lat_mean = {m: np.mean([lat[(m, s)] for s in
                        [r["scenario"] for r in grid if r["method"] ==
    m]])
            for m in ORDER}
lat_err = {m: np.mean([lat_ci[(m, s)] for s in
                       [r["scenario"] for r in grid if r["method"] == m
    ]])
           for m in ORDER}
baseline = lat_mean["No Validation"]

fig, ax = plt.subplots(figsize=(8.5, 5))
vals = [lat_mean[m] for m in ORDER]
errs = [lat_err[m] for m in ORDER]
bars = ax.bar(ORDER, vals, color=[COLOURS[m] for m in ORDER],
              edgecolor="black", linewidth=0.5, width=0.6,
              yerr=errs, capsize=5, error_kw={"elinewidth": 1.1})
for b, v in zip(bars, vals):
    ax.text(b.get_x() + b.get_width() / 2, v + 0.02, f"{v:.3f}",
            ha="center", va="bottom", fontsize=10, fontweight="bold")
ax.axhline(baseline, color="#9aa0a6", linestyle="--", linewidth=1.1)
ax.text(3.45, baseline + 0.01, f"baseline ({baseline:.3f})", ha="right"
    ,
        fontsize=8, color="#666")
for i, m in enumerate(["Nonce-Only", "Counter-Only", "Hybrid"], start
    =1):
    ov = lat_mean[m] - baseline
    ax.annotate(f"+{ov:.3f}", xy=(i, lat_mean[m]),
                xytext=(i, lat_mean[m] + 0.07), ha="center", fontsize
    =8,
                color="#444", arrowprops=dict(arrowstyle="-", color="#
    bbb", lw=0.8))
ax.set_ylabel("Average validation latency (microseconds) +/- 95% CI")
ax.set_ylim(0, max(vals) + 0.18)
ax.set_title("Mean Per-Message Validation Latency by Method\n"
             "All sub-microsecond; ordering among protected methods is
    within noise")
for s in ("top", "right"):
    ax.spines[s].set_visible(False)
ax.grid(axis="y", linestyle=":", alpha=0.5)
fig.tight_layout(); fig.savefig(os.path.join(HERE, "fig4_latency_v2.png
    "), dpi=200, bbox_inches="tight")
plt.close(fig)


# ============================================================================
# FIGURE 5 : capability -- scenarios neutralised (ASR <= 1%)
# ============================================================================
neutralised = {m: sum(1 for s in scen if (asr[(m, s)] or 0) <= 1.0) for
    m in ORDER}
fig, ax = plt.subplots(figsize=(8, 4.8))
vals = [neutralised[m] for m in ORDER]
```

```
bars = ax.bar(ORDER, vals, color=[COLOURS[m] for m in ORDER],
              edgecolor="black", linewidth=0.5)
for b, v in zip(bars, vals):
    ax.text(b.get_x() + b.get_width() / 2, v + 0.08, f"{v} / 7",
            ha="center", va="bottom", fontweight="bold")
ax.set_ylabel("Attack scenarios neutralised\n(ASR <= 1%)")
ax.set_ylim(0, 7.8)
ax.set_title("Security Capability by Attack Coverage\n"
             "(number of the 7 attack scenarios where ASR <= 1%)")
for s in ("top", "right"):
    ax.spines[s].set_visible(False)
ax.grid(axis="y", linestyle=":", alpha=0.5)
fig.tight_layout(); fig.savefig(os.path.join(HERE, "fig5_capability_v2.
   png"), dpi=200, bbox_inches="tight")
plt.close(fig)

print(f"Generated 5 figures in:\n  {HERE}")
```

Listing 6: Plot Graph.py

Glossary

ASR (Attack Success Rate) The percentage of replayed messages that the validation mechanism incorrectly accepts as legitimate. Lower is better; ASR + DR = 100%.

Counter A sequence of number attached to each transmitted message. A receiver accepts a message only if its counter is newer than the last accepted value (typically within a forward window). Used to defeat ordinary replay.

Desynchronisation A condition in which a legitimate message is jammed in transit and never reaches the receiver, leaving the receiver's stored freshness state behind the sender's. Exploited by RollJam-style attacks.

DR (Detection Rate) The percentage of replayed messages that the validation mechanism correctly rejects. Higher is better.

ECU (Electronic Control Unit) The embedded controller inside a vehicle that receives commands from the key fob and operates the actuator (door lock, ignition, etc.). In this project the ECU is the receiver in the one-way channel.

FRR (False Rejection Rate) The percentage of genuine messages that the validation mechanism wrongly rejects. Lower is better.

Hybrid validation The proposed two-step freshness check evaluated in this project: a message is accepted only if its nonce is unseen *and* its counter is inside the forward acceptance window.

Key fob The hand-held transmitter carried by the vehicle owner. In this project it is modelled as a one-way, battery constrained sender that issues commands such as `UNLOCK`, `LOCK` and `START`.

Latency (validation latency) The per-message processing time of the validation logic, measured in microseconds. Distinct from end-to-end response time, which includes the physical actuator.

Nonce A random number used once, attached to each transmitted message. A receiver accepts a message only if its nonce has not been seen before. Used to defeat replay by uniqueness.

One-way channel A communication setting in which the sender transmits to the receiver but the receiver does not transmit back. RKE channels are one-way, which rules out challenge-response defences.

Replay attack An attack in which an adversary captures a legitimate transmitted message and re-transmits it later to gain unauthorised access. The attack uses a genuine message; defeating it requires a freshness check, not stronger encryption.

Reset (volatile reset) A fault condition in which the receiver loses power. Volatile memory (the nonce list) is cleared and in a non-volatile memory (the stored counter) is preserved. Defeats nonce-only protection.

RKE (Remote Keyless Entry) The wireless system that lets a user lock, unlock or start a vehicle using a key fob. The threat model of this project.

Rolling code A counter-based RKE scheme in which the transmitted code changes with every press, accepted by the receiver only within a forward window. Used by KeeLoq and successor designs.

Rollback attack An attack that forces the receiver's stored counter backwards, making previously captured codes acceptable again. Defeats counter-only protection.

Validation Module The component of the receiver that runs the chosen freshness check (no-validation, nonce-only, counter-only or hybrid) on every incoming message.