IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 November 2025, accepted 7 December 2025, date of publication 16 December 2025,
date of current version 22 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3644958

RESEARCH ARTICLE

CARBUROS: A Formally Verified Authentication Protocol for V2X Communications

MARCO DE VINCENZI[1,2], CHIARA BODEI[2], GABRIELE COSTA[3], AND ILARIA MATTEUCCI[1]
[1]Institute for Informatics and Telematics (IIT), CNR, 56124 Pisa, Italy
[2]Department of Computer Science, Universitá di Pisa, 56127 Pisa, Italy
[3]IMT School for Advanced Studies, 55100 Lucca, Italy

Corresponding author: Marco De Vincenzi (marco.devincenzi@iit.cnr.it)

This work has been partially supported by the Italian National Recovery and Resilience Plan (PNRR) project Securing sOftware Platform (SOP) and received funding from the Three-Year Plan for Research 2022–2024 (PTR 22–24), Action P2.01 (Cybersecurity). Additional support was provided by SEcurity and RIghts in CyberSpace (SERICS), Project PE00000014, under the National Recovery and Resilience Plan (NRRP) of the Italian Ministry of University and Research (MUR), funded by the European Union – NextGenerationEU (EU-NGEU).

ABSTRACT The growing adoption of Vehicle-to-Everything (V2X) communications in Intelligent Transportation Systems (ITS) demands robust authentication protocols to ensure secure and reliable interaction among road actors. In this context, Multi-Factor Authentication (MFA) can serve as an important building block; however, integrating MFA into vehicular environments remains nontrivial. Traditional MFA verifies digital credentials but typically does not confirm physical presence. In ITS, authentication often requires binding digital identities to a vehicle's actual position and timing. To address this gap, this work presents CARBUROS, an MFA protocol specifically designed for ITS scenarios. It introduces a dual-channel mechanism that combines a standard digital communication channel with a physical proximity-based channel, ensuring that authentication reflects real-world co-location in space and time. This approach strengthens security by linking digital credentials to physical context. CARBUROS was formally modeled and verified using ProVerif, an automatic cryptographic protocol verifier. The analysis shows that the protocol withstands threats such as unauthorized access, identity spoofing, and functional disruption. The results confirm that CARBUROS preserves authentication and confidentiality under the modeled adversarial conditions while maintaining lightweight communication and computational overhead suitable for deployment in ITS environments.

INDEX TERMS Formal verification, multi-factor authentication, ProVerif, security protocols, vehicle-to-everything (V2X), vehicle-to-infrastructure (V2I).

I. INTRODUCTION

As new vehicular paradigms, such as Software-Defined Vehicles (SDVs), are gradually integrated into Intelligent Transportation Systems (ITSs), the need for secure and reliable communication increases accordingly [1]. In particular, ITS environments require strong guarantees of security properties such as vehicle authentication, message integrity and service availability. This is particularly evident in Vehicle-to-Everything (V2X) communications, where safety-critical operations rely on the trustworthy and timely exchange of data [2], supported by robust communication channels that span both in-vehicle and edge platforms [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Lorenzo Mucchi.

As SDVs increasingly rely on these channels for autonomous decision-making, failing to enforce these security guarantees could lead to service disruption or manipulation of traffic behavior, ultimately compromising the reliability of future transportation systems [4].

Authentication protocols are critical to ensure trust in V2X communication, particularly to prevent impersonation, spoofing, and unauthorized access. These threats can have safety implications. For example, an attacker who impersonates an emergency vehicle could abuse signal preemption systems, disrupting traffic, or creating hazardous conditions. Similarly, unauthorized access to restricted areas could pose physical security risks. Although authentication alone does not prevent all cyberattacks, it is a foundational component of ITS security architectures, as reflected in standards such

as IEEE 1609.2.1 [5] and ISO 21177 [6]. Due to the strict correlation between security and safety in Cyber-Physical Systems (CPSs), even a single breach can have severe consequences [7], [8], [9]. For example, in vehicle ramming attacks, an adversary that gains control of a non-autonomous or autonomous vehicle can pose serious safety risks [10].

Security standards in ITSs, such as Public Key Infrastructure (PKI) [2], ensure the confidentiality and integrity of digital communications. As a consequence, PKI-based credential management is already a critical element in major V2X security standards (including those adopted by IEEE and ETSI), and is actively deployed in pilot systems such as Security Credential Management System (SCMS) in the USA [11] and C-ROADS in the EU [12]. However, many existing protocols do not adequately account for the physical environment in which vehicles operate [5]. In V2X the physical and digital layers interact closely, yet PKI-based solutions alone are often not integrated with mechanisms that leverage physical-layer information to enhance overall trust and resilience.

Traditional authentication schemes are typically focused on digital-only identities and interactions, whereas vehicles are mobile, physical entities situated in space and time. This fundamental distinction calls for new forms of authentication evidence and corresponding formalization, capable of binding digital credentials to physical presence. As demonstrated in recent research [13], [14], this gap leaves systems vulnerable to adversaries who can exploit the misalignment between logical credentials and physical reality. Interestingly, also the IEEE 1609.2.1 standard acknowledges this issue, but without providing a rigorous definition of a physically secure environment [5].

In this context, a physically secure session is generally defined as a communication session in which security is guaranteed only if both endpoints, e.g., a vehicle and a Roadside Unit (RSU), are co-located within a local area. For example, the communication may rely on short-range visual, infrared, or mmWave signals that are inherently limited in spatial reach, ensuring that only entities in physical proximity can exchange messages. However, without a precise characterization of these elements, one cannot verify whether the security goals of a protocol are achieved.

The present study builds on the approach proposed by Suo et al. [15], who introduced a two-factor authentication scheme employing a Line-of-Sight (LOS) side channel, to mitigate impersonation attacks by roadside adversaries with compromised credentials. Their subsequent real-world implementation [13], [14], progressively evolved into a multi-channel authentication mechanism and was successfully tested in a real vehicle. However, while these experimental results demonstrate feasibility, the underlying communication protocol had not been formally specified and analyzed, and aspects such as message structure, sequencing, and attacker models still required formal treatment.

To address these issues, CARBUROS, a vehicular Multi-Factor Authentication (MFA) protocol, was designed and formally verified, providing a framework for physical authentication of vehicles within the ITS ecosystem. CARBUROS implements a dual-channel authentication mechanism that combines a conventional digital communication channel with a physical proximity-based channel. The proximity channel, requiring short-range interaction, is designed to confirm the physical presence of the communicating entity, thereby binding digital credentials to spatial context. Overall, it provides a formally verified integration of physical-layer evidence into a multichannel vehicular authentication protocol.

Beyond the original design, CARBUROS introduces explicit attacker modeling and symbolic verification, extending the previous proof of concept [15] into a formally verified and implemented multichannel protocol that integrates physical-layer evidence.

The main contributions are summarized as follows.

- **Protocol design:** CARBUROS was conceived as a two-channel MFA protocol specifically tailored for Vehicle-to-Infrastructure (V2I) scenarios. In this configuration, the vehicle initiates authentication toward the infrastructure, while the final decision is made by a trusted authority, typically the Registration Authority (RA), based on information provided by the RSU, which operates as the local verifier.
- **Formal verification:** CARBUROS was formally modeled and verified using ProVerif [16], demonstrating that the protocol ensures key security properties, including the secrecy of authentication tokens and the successful completion of the authentication process, verified as reachability properties.
- **Extended attacker model:** Two categories of adversaries are considered. ''Remote attackers'' are defined as adversaries that are not co-located with the vehicle or the RSU and operate exclusively over wireless communication channels. This category includes traditional Dolev-Yao (DY) [17] adversaries with full control over the network infrastructure. In contrast, ''proximity-based adversaries'' are capable of observing or interfering with communication channels that require physical nearness, such as LOS optical links used for visual authentication. To capture both threat types, the standard DY model supported by ProVerif was extended to incorporate a notion of proximity and attacker capabilities specific to vehicular environments. In particular, two additional attackers were introduced: (i) a novel proximity-based adversary and (ii) an Attacker-in-the-Device (AitD), operating through a fully compromised device, as detailed in Section IV-B.

The remainder of this paper is structured as follows. Section II compares this work with related approaches in vehicular authentication. Section III provides the background, introducing symbolic modeling of protocols and their properties, as well as the ProVerif tool. Section IV designs the threat model and the different types of attackers. Section V describes the CARBUROS protocol and its message sequence, with a focus on security goals and

design choices. Section VI presents the formal modeling and verification of the protocol using ProVerif, highlighting the results obtained in various attacker scenarios. Section VII provides a discussion of the results, while Section VIII concludes the article and outline future research directions.

II. RELATED WORK

The literature related to CARBUROS can be grouped into four main research directions: (i) dual-channel and proximity-based authentication protocols, (ii) formal verification for vehicular and automotive systems, (iii) MFA models in the cyber-physical domains, and (iv) complementary advances in intelligent and adaptive security.

A. DUAL-CHANNEL AND PROXIMITY-BASED AUTHENTICATION PROTOCOLS

As stated in the Introduction, CARBUROS builds on the concept introduced by *Suo et al.* [15], who addressed impersonation and message forgery by introducing LOS communication as a secondary authentication factor. Their method required vehicles to respond to challenges received over the Non-Line-of-Sight (NLOS) channel via a directional LOS channel, making attacks from stationary adversaries more difficult. A key innovation was the use of vehicle motion to impose physical restrictions that validate identity. *Dwyer et al.* [14] implemented this idea through QR codes transmitted over the LOS channel and detected by roadside cameras with deep learning models. Later, *De Vincenzi et al.* [13] extended this approach using real vehicles and a neural network to interpret responses via the visual channel, achieving over 96% accuracy in real-world tests.

Another dual-channel authentication scheme was proposed by *Alsoliman et al.* [18], who combined Optical Camera Communication (OCC) and Radio Frequency (RF) channels to authenticate and localize message transmitters in autonomous vehicles. By transmitting synchronized nonces over both channels, the system countered location spoofing and identity duplication threats. CARBUROS expands upon these principles, integrating additional protections through formal verification and attacker modeling.

In summary, it is important to note that CARBUROS does not belong to a single, well-defined family of protocols. It incorporates elements from traditional cryptographic network protocols, multi-factor authentication schemes involving physical proofs, and distance-bounding mechanisms, yet none of these categories fully captures its scope, and no directly comparable vehicular MFA protocols exist, to the best of our knowledge.

B. FORMAL VERIFICATION OF VEHICULAR PROTOCOLS

Formal methods are increasingly used to ensure the correctness and security of vehicular communication protocols. In the literature different tools have been used to analyze security protocols. The most popular ones are ProVerif [16], Tamarin [19], AVISPA [20], and Scyther [21]. Based on [22], Table 1 summarizes the main features of the four tools and compares them with respect to the analysis of security protocols.

Referring to the existing literature, ProVerif [16] and Tamarin [19] are more relevant in the analysis of vehicular protocols. Indeed, *Bruni et al.* [23] conducted one of the first formal analyses in this domain by evaluating the MaCAN protocol using ProVerif. Their findings confirmed MaCAN's security while maintaining compatibility with the constrained bandwidth of the Controller Area Network (CAN) bus.

In 2020, *Lauser et al.* [24] analyzed AUTOSAR's Secure Onboard Communication (SecOC) module using the Tamarin prover [19], focusing on authenticity, group membership, and message freshness. *Krichen* [25] later highlighted the role of formal verification techniques, such as theorem proving and abstract interpretation, in the automotive development lifecycle, emphasizing their complementarity with testing and fuzzing approaches. *Bodei et al.* [26] analyzed the CINNAMON protocol [27], [28] using Tamarin, showing that it provides confidentiality, unlike SecOC, which lacks encryption. This work reinforced the importance of formal tools such as Tamarin and ProVerif in automotive cybersecurity.

Costa et al. [29] explored physical-layer security in symbolic models, and used ProVerif to verify confidentiality and integrity properties for protocols based on watermarking and jamming. Their results demonstrated that these energy-efficient mechanisms can complement traditional cryptographic techniques, especially in low-power vehicular or sensor networks.

To formally analyze CARBUROS, ProVerif was selected because, compared to other existing tools, it offers the highest degree of automation and supports all targeted properties, including secrecy and authentication. As the analysis focuses on verifying specific security properties, ProVerif enables to automatically check their satisfiability, including secrecy and reachability of the final state. Other tools, such as Scyther and Tamarin, are primarily oriented toward identifying potential attack traces. In contrast, ProVerif is easier to customize and enables the definition of different scenarios and attacker models against which the CARBUROS protocol can be verified. Future analyses could complement this approach by employing Tamarin to incorporate time-dependent and fairness properties.

C. MULTI-FACTOR AUTHENTICATION IN CYBER-PHYSICAL SYSTEMS

Research on MFA has expanded beyond digital systems to include cyber-physical environments. *Jacomme et al.* [30] modeled MFA schemes such as Google 2-Step Verification and FIDO U2F using applied pi calculus and ProVerif, accounting for compromised devices and phishing attacks. While their analysis addressed human-centric systems, CARBUROS introduces a vehicular adaptation of MFA that binds authentication to both digital credentials and physical presence.

TABLE 1. Comparative overview of verification tools and protocol features.

Tool	Verification Method	Formal Language	Attack Model	Strenght
ProVerif	Symbolic analysis based on applied pi-calculus	Logical deduction (Horn clauses)	Dolev–Yao intruder with full network control and infinite sessions	Automated, supports secrecy, authentication, privacy, and equivalence proofs
Tamarin Prover	Rule-based rewriting system with temporal logic	Symbolic rewriting plus temporal reasoning	Dolev–Yao intruder, can also model stateful compromises (e.g. long-term or ephemeral key leakage)	Handles mutable state, key updates, and compromise scenarios; powerful temporal reasoning
AVISPA	Model checking using HLPSL and multiple back-ends	State exploration (model checking)	Dolev–Yao intruder with bounded sessions and explicit message interception/modification	Supports standard protocols (TLS, IKE, Kerberos); produces clear attack traces
Scyther	Symbolic analysis for finite sessions	Symbolic state exploration	Dolev–Yao intruder (active adversary controlling communication channels)	Intuitive, fast, clear attack traces

Recent studies have applied similar approaches in Intelligent Transportation Systems (ITS) and Electric Vehicle (EV) domains. *Van Aubel and Poll* surveyed EV charging protocols, highlighting gaps in driver authentication [31]. *Mookherji et al.* proposed a secure ultra-fast authentication protocol for charging stations [32], while *Hamdare et al.* [33] developed a defense framework against vulnerabilities in Open Charge Point Protocol (OCPP) communications. *Ying et al.* [34] provided a broad survey of authentication and physical-layer techniques for V2X, and *Tanyıldız et al.* [35] addressed intrusion detection in EV communication networks.

In contrast, CARBUROS unifies physical presence verification and digital credential validation within a single, formally verified multi-factor process. Its dual-channel architecture enables authentication not only for static radio-based services but also for dynamic inductive charging and mobile V2X scenarios. This can represent a step toward achieving scalable and trust-based authentication across vehicular and edge ecosystems.

D. COMPLEMENTARY ADVANCES IN INTELLIGENT AND ADAPTIVE SECURITY MECHANISMS

Recent advances in intelligent threat detection further complement CARBUROS's goals. *Qiqieh et al.* [36] proposed a swarm-optimized framework using Harris Hawks Optimization and Support Vector Machines for multi-domain intrusion detection. Similarly, *Alzubi et al.* [37] developed an optimized intrusion detection system for fog and edge environments using Denoising Autoencoders. These approaches emphasize adaptive and resource-efficient protection, resonating with CARBUROS's design philosophy that combines contextual verification and MFA in distributed vehicular systems.

III. BACKGROUND

Symbolic models are widely used to verify the security properties of communication protocols. In this abstraction, messages are represented as terms in a formal algebra equipped with cryptographic constructors and destructors, rather than as raw bitstrings. Cryptographic primitives are treated as ideal closed boxes, and their behavior is specified through equations or inference rules that may include algebraic properties when needed.

Within this framework, the adversary follows the classical DY model: it controls the network, intercepts all messages, forges new ones using available constructors, and applies destructors to extract information. The evolving knowledge of the attacker is represented as a set of terms and its deduction capabilities are formalized through a proof system that specifies which messages can be derived from previously acquired knowledge.

A. AUTOMATED PROTOCOL VERIFICATION AND ProVerif

This symbolic foundation underlies several automated verification tools, most notably ProVerif [16], which is based on the applied pi-calculus [38]. This process algebra extends the standard pi-calculus with cryptographic operations and provides a convenient formal language for specifying protocols and their security properties. ProVerif translates the protocol into a set of Horn clauses and expresses the security goals as derivability queries. It then applies resolution to determine whether each fact can be derived. If a fact is not derivable, the corresponding property holds; if it is derivable, ProVerif reports a potential violation or attack.

B. ProVerif SPECIFICATION

A ProVerif specification typically consists of two parts. The first defines the communication channels, the message structure, and the cryptographic primitives, together with the equations that formalize their algebraic behavior. The second part specifies the behavior of protocol participants as parameterized processes in the applied pi-calculus, which can generate fresh terms, send and receive messages, and execute concurrently.

Terms are typed, with predefined types including `channel` for communication channels, `bool` for boolean

values and `bitstring` for binary data. Additional types can be declared by users. The syntax `channel c.` stands for `free c: channel`. By default, free names are known to the attacker. Names not known by the attacker must be declared as private: `free n: t [private].` Constructors (function symbols) are defined using the format `fun f(t1,...,tn): t.`, where `f` is a constructor of arity n, `t` is its return type, and $t_1, \ldots, t_n$ are the types of its arguments. Relationships between cryptographic primitives are expressed through destructors, which are used to manipulate terms created by constructors. For instance, the representation of the asymmetric (public-key) encryption scheme in ProVerif is as follows:

```
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey;
adec(aenc(m, pk(k)), k) = m.
```

with two constructors: (i) `pk(skey):pkey` for deriving the public key from a secret key, and (ii) `aenc(bitstring,pkey):bitstring` for encrypting a message using a public key. The subsequent reduction rule for the decryption destructor states that for all messages `m` (of type `bitstring`) and secret keys `k` (of type `skey`), decrypting a message encrypted with the public key `pk(k)` using the corresponding private key `k` yields the original message `m`. Constructors and destructors can also be declared private.

C. SECURITY PROPERTIES

ProVerif can address reachability properties, correspondence assertions, and observational equivalence properties. Reachability properties can be exploited to evaluate the secrecy of terms. To determine whether a term `M` remains confidential in a model, the query `query attacker(M).` is included in the input file, in which `M` is a ground term (without destructors) that may be private and initially unknown to the attacker. If the attacker cannot derive M, the term is proven secret.

Correspondence assertions are used to define relationships between protocol events in the following form: if a principal executes an event `e` (e.g., completing a protocol session), then another principal must have executed a corresponding event `e'` (e.g., initiating that session).

Events can include arguments, which allows for the analysis of relationships between their values. To use these assertions, processes are annotated with events that mark significant protocol stages, without altering the protocol's behavior.

IV. ATTACKER MODEL

In this section, a detailed overview of the reference threat model is provided, which highlights the goals and technical capabilities of the attacker.

A. ATTACKER'S GOALS

As in many authentication protocols, the attacker aims to subvert the correct identification of legitimate users. In particular, this work distinguishes the following three main goals.

1) UNAUTHORIZED ACCESS

In this scenario, the attacker attempts to authenticate to the system without possessing valid credentials. Under this working assumptions, this can involve a tampered or even fictitious vehicle that should not be allowed access to the road section. From the protocol's perspective, "the correct authentication" or "a successful attack" occurs when the attacker can properly complete the protocol message flow, either independently or by hijacking an ongoing session of a legitimate user.

2) IDENTITY SWAP

An identity swap occurs when two users successfully complete the authentication procedure, but the system incorrectly assigns each user the other's digital identity. Even when both users are legitimate, such a mix-up can lead to unintended behavior, in particular when authentication feeds an authorization system. For instance, an identity swap could be exploited to carry out privilege escalation or to trigger a confused deputy attack. In the scenario described in this work, the attacker may achieve this by interfering with two authentication flows, e.g., by redirecting some messages or scrambling their order.

3) PROTOCOL FAILURE

The third case involves an attacker who aims to prevent a legitimate user from successfully completing the authentication process.

The focus is on scenarios in which an attacker forces the protocol to enter a *faulty state*, for example, when the authentication server repudiates a legitimate user at the end of the protocol. More generally, protocol failures can also be associated with execution interruptions, where runs that never reach a terminal state. Such cases can occur when communication channels are physically disrupted or intentionally blocked by the attacker. However, these situations fall outside the scope of the present work.

B. ATTACKER'S CAPABILITIES

Defining the reference attacker model in terms of capabilities is non-trivial, particularly in the context of protocols such as CARBUROS, which involve multiple communication channels with distinct security properties. This complexity is characteristic of strong authentication protocols that integrate both digital and physical-layer interactions. To encompass a spectrum of realistic threat scenarios, three attacker types were defined, each corresponding to a different position or level of access within the system.

In this work, Umarell and AitD are introduced as complementary attacker classes to bridge the gap between the abstract network omnipotence of the DY model and the spatially or logically constrained adversaries encountered in realistic vehicular environments.

The Umarell model captures localized interference, such as jamming, partial eavesdropping, or selective frame loss, that are difficult to express through DY assumption of global network control. Unlike DY, Umarell is bound by physical proximity and cannot arbitrarily inject or alter messages across the network. This makes it crucial for modeling adversaries that exploit optical, radio, or LOS channels within a limited range. The model reflects realistic cases such as an external observer positioned near an intersection attempting to disrupt or shadow a visual component like a traffic light without access to the broader vehicular network [38].

Conversely, AitD embodies the growing risk of internal compromise within connected vehicles and infrastructures. It models adversaries that gain partial or full control of a legitimate component (e.g., a tampered Electronic Control Unit (ECU), an infected RSU, or a compromised vehicle node) while still possessing valid cryptographic credentials. Such insider attacks cannot be represented under the DY model, which presumes honest endpoints. By incorporating AitD, CARBUROS can be evaluated under conditions where an apparently trusted entity violates the protocol's assumptions from within, thus testing resilience against insider threats, key leakage, or firmware tampering.

Together, these models extend formal verification beyond the idealized DY abstraction, enabling the analysis of proximity-bounded and insider threats that are increasingly relevant in multi-channel V2X authentication schemes like CARBUROS.

In the following, these attacker types and their alignment with the working assumptions are outlined.

1) DOLEV-YAO

The DYs attacker is widely considered the standard adversary for the verification of network security protocols, as in the ProVerif tool. The main reason is that its strong capabilities are generally considered a safe over-approximation of real-world attackers. Consequently, DY-proof protocols do not fall against weaker attackers.

Furthermore, DY has unlimited memory, i.e., it can reuse any piece of information independently from when it was acquired, and unlimited computational power, i.e., it can instantly solve problems for which an efficient algorithm exists. Nevertheless, DY operates under the perfect cryptography assumption, meaning it cannot break cryptographic primitives, e.g., it can only decrypt and sign messages if it has previously disclosed the needed keys. As remarked by some authors, DY may not be fully adequate for analyzing certain security protocols. For instance, in [39] the authors show that DY is overly strong against multi-factor authentication protocols and, at the same time, it may miss some capabilities that real attackers have, e.g., as in social engineering attacks.

2) UMARELL

To capture proximity-based threats not modeled by existing attacker classes, a novel attacker type, termed *Umarell*,[1] was introduced.

This name stands for an agent that observes and possibly interferes with a target communication channel. The main characteristic of an Umarell attacker is its proximity to the channel, either physically or via some remotely controlled device. In terms of capabilities, Umarell is much weaker than the DY attacker. In fact, it can interfere only with a subset of existing channels, such as those involved in the communication field. Moreover, unlike DY, Umarell cannot deliver messages. Hence, for each transmission, it can decide to eavesdrop on or delete the message, but not both. Furthermore, it lacks the ability to transmit. Existing attacker models typically assume either complete control of the network (DY) or full compromise of a participant. However, neither captures localized attackers that can observe or disrupt communication only within a physical range, a relevant case in V2I and proximity-based authentication. Umarell has been introduced to fill this gap and to model realistic, physically bounded adversaries.

3) ATTACKER-IN-THE-DEVICE

The AitD model refers to an adversary that has fully compromised one of the protocol's parties, e.g., a tampered vehicle under the attacker's control. In the literature, this kind of adversary is often named after a specific technology, e.g., Machine-in-the-Browser (MitB) or Machine-in-the-Mobile (MitMo) [40], [41]. AitD is an extremely powerful attacker, but is restricted to operate only through the compromised device. In particular, AitD has the same capabilities as a DY attacker on all channels connected to the compromised device. Furthermore, it has access to all secret information shared with the device, such as secret keys. Instead, channels that do not involve the controlled device remain beyond the AitD's reach. Although extremely powerful, AitD attackers can be partly mitigated in real systems through defenses such as secure boot, remote attestation, trusted execution, or hardware security modules, which aim to limit the impact of a compromised device.

V. CARBUROS

The purpose of CARBUROS[2] is to authenticate vehicles as they move across different sections of the road. It is assumed that CARBUROS operates within a broader ecosystem in which vehicular services are provided.

Its modular and adaptable architecture makes it suitable for integration with various proximity-verification chan-

[1] A neologism for a retired person who spends time observing construction sites, especially roadworks. See https://en.wikipedia.org/wiki/Umarell

[2] The name playfully references the well-known Kerberos authentication protocol.

nels, such as optical, infrared, or mmWave. This work demonstrates CARBUROS based on the experimental setting described in [13], which employs optical channels.

A. ARCHITECTURE DESIGN

Figure 1 illustrates the CARBUROS communication system, which involves three main entities: the RA, the RSU, and the vehicle V. The RA plays the role of a central and authoritative system, it manages the validation and issuing of the certificates. For these reasons, only the RA is always assumed to be a trusted entity within CARBUROS.

The proposed architecture complements and builds on a PKI system, such as those specified in IEEE 1609.2.1 [2].

This design aligns with ongoing standardization efforts under IEEE 1609.2.1 and ISO 21177-2024 [42], both of which define secure authentication for V2X communications. CARBUROS extends these frameworks by introducing a formally verified, proximity-aware layer that can interoperate with existing PKI infrastructures without violating compliance requirements.

Consequently, it was assumed that each CARBUROS party was provisioned with appropriate cryptographic keys, which were managed over time via key-distribution services.

In addition, before authentication, it is assumed that each vehicle completes an enrollment phase during which it is registered with the road infrastructure. As in other transportation infrastructures, the enrollment phase assigns each vehicle a unique identifier, called a *squawk* [43], [44], a term derived from the aviation jargon.

In CARBUROS, the squawk identifier is generated and assigned by a Trusted Authority (TA) to represent a vehicle within a specific area or time-limited session. The squawk serves as a pseudonymous identity that allows the vehicle to participate in authentication exchanges without disclosing its permanent credentials. Only the TA maintains the internal mapping between the squawk and the real vehicle identity, ensuring that RSUs or other entities cannot correlate multiple sessions belonging to the same vehicle. This design provides controlled anonymity, allowing the squawk to act both as a temporary identifier and as a session reference during protocol execution. Since each squawk is unique, short-lived, and issued only by the TA, external observers cannot track vehicles across regions or time periods. This mechanism is compatible with existing vehicular privacy standards, such as pseudonym certificate rotation in IEEE 1609.2, allowing CARBUROS to combine strong authentication with privacy-preserving identity management.

After the enrollment phase, the authentication process can take place between a vehicle and the road infrastructure, such as an RSU equipped with sensors.

CARBUROS adopts a dual-channel architecture strengthening security by leveraging the complementary benefits of both [15]:

- *Remote channel*: a reliable channel with sufficient bandwidth to enable the exchange of complex data between the vehicle and the RSU, operating without requiring physical proximity.
- *Physical proximity-based channel*: a channel of limited bandwidth that is meant to be accessible only by vehicles located near the RSU.

The two types of channels connect vehicles with the local RSU as depicted in Fig. 2. As mentioned above, CARBUROS is designed to be independent of the kind of channel technology. However, in the following, the LOS optical channel from [13] is employed as a representative realization. In this setting, authentication involves visual signals exchanged between a vehicle (through its headlights) and an RSU.

LOS solutions are already widely used in ITS, for example in speed enforcement, license plate recognition and traffic management. Leveraging LOS communication for authentication offers the key advantage of verifying physical proximity. To ensure robustness in possible adverse and dynamic real-world conditions, CARBUROS may incorporate fallback mechanisms that maintain service continuity even when LOS authentication is temporarily unavailable. For example, if the LOS channel is obstructed, due to traffic, infrastructure, or weather, the system can automatically re-initiate the challenge after a timeout or redirect the authentication process to a nearby RSU. CARBUROS modular design also enables flexible implementation of the LOS channel using not only visible light, but also alternative technologies such as infrared or mmWave radar. This extensibility opens new directions for research and optimization in scenarios with potential obstructions or low visibility.

In the reference implementation, the RSU continuously monitors the physical channel for a valid optical challenge response within a defined time window (typically 2–3 s). If the expected flash sequence is not detected or decoding fails due to obstruction, the RSU triggers a timeout event signaling a LOS blockage. This event initiates a controlled fallback sequence: the RSU requests a retransmission of the challenge over the same channel, and, if the failure persists, the vehicle is redirected through the remote channel to a nearby RSU as designed by the RA. The fallback process is securely coordinated by the RA, which maintains session continuity and ensures that the authentication context (including the squawk identifier) remains valid.

The RSU is equipped with sensors, such as a camera, and is connected to the RA through a traditional network infrastructure, e.g., a Wide Area Network (WAN).

Nevertheless, the protocol remains *channel-agnostic*, meaning it is independent of specific communication media, technologies, and application domains. For instance, the first channel, used for standard data exchange, is wireless and can be implemented using a variety of communication technologies. In contrast, the second channel can rely on alternative methods, such as visual communication (as in the reference implementation) or electromagnetic signals. This design flexibility ensures that CARBUROS can adapt

to diverse deployment contexts without altering its core logic or security guarantees. See Subsection V-D for more details.

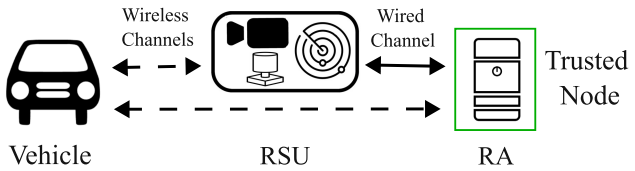


FIGURE 1. CARBUROS architecture with its main entities: RA, RSU, and vehicles.

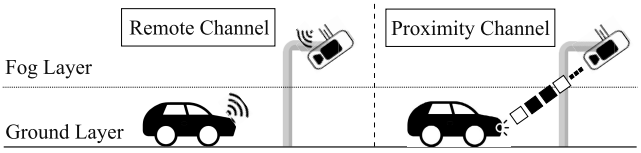


FIGURE 2. CARBUROS two-channel schema, where the second channel can be a physical proximity-based, such as a LOS channel [13], [14].

B. MESSAGE SEQUENCE

The sequence diagram of CARBUROS (Fig. 3) illustrates the temporal order of events and the interaction flow among entities. Complementarily, the protocol is also specified using Alice–Bob notation [45], which emphasizes the message structure and the cryptographic operations involved.

$$1.V \rightarrow RA : \{squawk, loc\}_{TLS}$$
$$2.RA \rightarrow V : \{RSUdata\}_{TLS}$$
$$3.V \rightarrow RSU : \{squawk, bitmask\}_{pk_{RSU}}$$
$$4.RSU \rightarrow V : chall \oplus bitmask$$
$$5.V \dashrightarrow RSU : chall \text{ [optical channel]}$$
$$6.V \rightarrow RA : \{squawk, chall\}_{TLS}$$
$$7.RSU \rightarrow RA : \{squawk, chall\}_{TLS}$$
$$8.RA \rightarrow V : \{[token]_{k_{RA}}\}_{TLS}$$

The protocol consists of eight messages grouped into three phases: *setup* (steps 1-2), *challenge* (steps 3-5) and *check* (steps 6-8). The detailed operation of each phase is described below.

1) SETUP STEPS

The setup phase establishes the initial communication between the vehicle and the road infrastructure, enabling the identification of the appropriate RSU. During this phase, the vehicle transmits its identifier (*squawk*) and current location to the RA, which in turn provides the connection details of the designated RSU.

1) The vehicle V starts the protocol by sending its *squawk* and location (*loc*) to the RA over a Transport Layer Security (TLS)-protected channel.
2) Over the same TLS channel, RA replies with the connection details of the RSU associated with *loc* (denoted as RSUdata), such as its network address.

2) CHALLENGE STEPS

The challenge phase verifies both the vehicle's proximity to the RSU and the vehicle's authenticity through a two-step interaction. In this phase, RSU presents a randomized challenge that the vehicle must respond correctly through the proximity-based channel.

3) The vehicle generates a random, one-time bitmask *mask*. It then encrypts its squawk and *mask* using the public key of the RSU and sends the resulting message to the RSU.
4) The RSU decrypts the message and generates a fresh value *chall*, a random bitstring of the same length as *mask*. Then, the RSU computes the XOR (exclusive OR) between *chall* and *mask* and sends the result back to the vehicle.
5) Using the physical proximity-based channel (dashed arrow), the vehicle responds to the challenge *chall*. In [13], this is achieved through the vehicle's headlights, which emit a flash sequence encoding the challenge response. The RSU's camera detects and interprets this visual signal, verifying that it matches the expected sequence.

3) CHECK STEPS

The check phase terminates the verification process by cross-referencing the vehicle response across dual-channel and two-actor communication. Upon successful validation, the vehicle is issued an authentication token that confirms its legitimacy.

6) The vehicle sends its squawk and the response to the challenge to the RA over a TLS-protected channel. This allows the RA to directly validate the signature of the response to challenge received from the vehicle.
7) The RSU forwards squawk and *chall* to the RA, providing confirmation of the vehicle response received over the physical channel. This step enables RA to detect any inconsistencies or tampering by cross-referencing the responses received from both communication channels and both actors.
8) Upon successful validation of the responses, the RA generates an authentication token. The token is encrypted with RA's private key and addressed to the vehicle's public key, ensuring secure and verifiable delivery.

To mitigate potential desynchronization attacks, i.e., attempts to exploit timing or coordination mismatches between the remote and physical channels, CARBUROS enforces tight coupling between both communication paths. Each challenge–response cycle is bound to a unique session identifier (the *squawk*) and synchronized nonce (*bitmask*) generated by the RSU. A strict time window is also enforced for message verification: if either the optical or TLS response exceeds this threshold, the authentication process is aborted and a re-synchronization request is issued. This mechanism prevents adversaries from desynchronizing the two channels while maintaining minimal impact on latency and throughput.

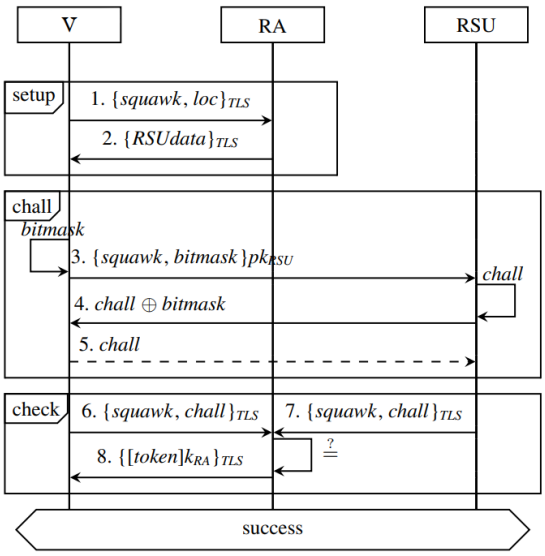


FIGURE 3. CARBUROS sequence diagram.

C. SECURITY GOALS

The primary goal of CARBUROS is to authenticate a vehicle to the road infrastructure. This authentication allows the vehicle to obtain a token that grants access to high-security services. The token serves as an authorization credential that the vehicle can use to access location-specific services, for example, reserved lanes, toll discounts, dynamic EV charging stations, or signal preemption privileges for emergency vehicles.

Without proper authentication, malicious or unauthorized vehicles could access restricted road services, disrupt traffic management systems, or trigger privileges (e.g., green-light preemption) fraudulently. The use of dual-channel MFA and the proximity-verified token ensures that the vehicle is not only legitimate (i.e., in possession of valid credentials) but also physically present at the service location, thereby strengthening both digital trust and physical safety in real-world V2I interactions. Authentication succeeds only after the vehicle completes a correct execution of the protocol and receives a valid token issued by the RA. Hence, the goal is achieved after message 8 (see Fig. 3). Also, this occurs only when RA receives the same squawk-*chall* pair from both the vehicle and the RSU (messages 6 and 7).

Regarding the attacker's goals outlined in Section IV, CARBUROS must ensure the following.

1) The authentication token is issued only to a legitimate vehicle that owns valid credentials (i.e., the squawk and the private key associated with pk_V) and interacts with the RSU associated with the location *loc*.
2) The authentication token is delivered to the same vehicle that initiated the session.
3) Whenever vehicle V receives a token from RA, it will be accepted and not repudiated by an honest RSU.

D. DESIGN CONSIDERATIONS

The design of CARBUROS considers various aspects and requirements to ensure an efficient and adaptable system for real-world vehicular scenarios.

The protocol employs a dual-channel approach to improve security where the second channel serves as a physical-proximity confirmation layer to ensure that an attacker must physically be present within the direct line of communication to compromise the system. This mitigates the vulnerabilities associated with the remote channel, which facilitates efficient data exchange. For remote communication, the TLS protocol is used to ensure confidentiality and integrity, as described in IEEE 1609.2 [2]. This combination of physical and remote channels and the adoption of standard encryption solutions achieve a balance between security and practicality in the given context.

To address the temporal constraints of vehicular traffic, the challenge–response phase was designed to involve only short message exchanges. This minimizes latency and ensures timely authentication, critical in dynamic traffic conditions. In addition, the protocol can also limit the number of challenge configurations (e.g., in [13] a limited number of bits to be flashed) to accommodate vehicular speeds and operational constraints, while maintaining security through unique one-time elements such as the challenge and the bitmask.

Furthermore, the system generates unique configurations in the time required for several vehicles to pass through the authentication zone, ensuring scalability in high-density traffic scenarios. Even when multiple vehicles are authenticated simultaneously by the same RSU, distinct challenges can be issued to each of them. As the authentication process is designed to be completed within a few seconds, a relatively small number of configurations can be sufficient to meet the system's requirements.

1) EVALUATION OF THE PHYSICAL CHANNEL

The feasibility and quantitative performance of the CARBUROS proximity channel were validated in the companion study [13], which provides a statistical evaluation of the optical communication process in real traffic-like conditions. In this setup, vehicles exchanged challenge–response messages with the RSU through headlight-based optical modulation, while the remote channel maintained the cryptographic link over TLS. The experiments were conducted under various ambient lighting and weather conditions to evaluate robustness against interference and environmental noise.

Results confirmed that the challenge–response exchange at the physical layer was consistently completed within 1–3 seconds, depending on the vehicle speed (8.3–16.6 m/s) and the distance from the RSU (up to 50 m). Each optical flash lasted approximately 0.15 s, resulting in an overall latency fully compliant with vehicular safety and authentication requirements. On embedded hardware (NVIDIA

Jetson Orin), message decoding exhibited an average latency of 1.6 ms, confirming negligible computational overhead for near-real-time deployment. The employed SlowFast Convolutional Neural Network (CNN) classifier achieved an average recognition accuracy of 96.6% under dynamic lighting and noise, outperforming both 3D-CNN (85.6%) and *You Only Look Once (YOLO)* based *Quick Response (QR)* code schemes (42–100%, distance-dependent). These outcomes demonstrate that the optical channel remains resilient to noise and interference even in outdoor conditions and that decoding can be performed reliably with minimal delay.

The experiments also confirmed the scalability of the approach in multi-vehicle scenarios: distinct optical challenges can be generated and processed in parallel by a single RSU without mutual interference. The flash reading model only requires an additional module to manage different regions of interest (ROIs), cropping individual vehicles before using the CNN to continue the pipeline as described in [13]. This confirms the protocol's capacity to support simultaneous authentications in dense-traffic environments. Although the physical tests were conducted in a controlled urban setting, the observed results confirm the soundness of the timing assumptions adopted in the formal model and their feasibility in real deployments. Extended statistical results and large-scale performance analyses are detailed in [13], while this work focuses on the symbolic validation and formal verification of security guarantees.

A promising direction for improving the physical channel involves the use of Visible Light Communication (VLC) by leveraging the vehicle's Light-Emitting Diodes (LEDs) to emit flashes at specific frequencies imperceptible to the human eye. This technique enables secure short-range communication without interfering with driver visibility or requiring additional hardware. Moreover, VLC naturally supports proximity constraints: its dependence on LOS and limited range make it suitable for applications where physical presence must be verified. The communication pattern can be modulated at higher frequencies, allowing faster and more robust transmissions compared to simpler on–off flashing.

Although the present evaluation focuses on an optical LOS implementation, CARBUROS is inherently channel-agnostic. The physical channel is abstracted at the protocol layer as a verifiable proximity factor, meaning that alternative modalities, such as RF, VLC, or ultrasound, can instantiate the same mechanism. For instance, RF-based variants could exploit received signal strength, time-of-flight, or angle-of-arrival measurements to enforce physical proximity, while hybrid optical–RF approaches can combine reliability with spatial precision. This flexibility allows CARBUROS to adapt to diverse vehicular environments without altering its security logic or formal verification assumptions.

VI. FORMAL VERIFICATION USING ProVerif

In this section, the formal modeling and verification of CARBUROS using ProVerif are presented.

A. MODELING OVERVIEW

Since ProVerif is a model checker for security protocols that assumes the DY attacker model, its application to CARBUROS is not straightforward. Two distinctive aspects must be addressed: the threat model and the presence of a physical communication channel. To properly capture these characteristics, the following four modeling scenarios were introduced, as summarized in Table 2.

The inclusion of the Umarell and AitD attacker types extends the symbolic analysis beyond the classical DY model. While DY provides a powerful abstraction for full network control, it cannot represent attacks constrained by distance, visibility, or insider compromise. The Umarell attacker allows us to analyze the resilience of CARBUROS against localized, proximity-based interference on the physical channel, whereas the AitD scenarios evaluate the impact of compromised participants on the system's trust assumptions. By combining these models, the formal verification captures a broader and more realistic spectrum of threats relevant to V2X deployments.

- **S1.** A standard DY attacker attempts to violate the CARBUROS security goals (see Section V-C). The attacker has full control over the network infrastructure (represented by solid lines), including all remote channels, but cannot interfere with the physical channel (shown as dashed lines). This scenario features a roadway with two RSUs and two vehicles.
- **S2.** Scenario S1 is extended by introducing the Umarell attacker, who is capable of interfering with the physical channels.
- **S3.** The AitD attacker fully controls one vehicle and consequently all the channels connected to it. This scenario considers a single RSU.
- **S4.** Scenario S3 is modified by giving AitD control over the RSU, rather than the vehicle. As in S3, the attacker also controls all channels passing through the compromised device.

The cases outlined above are claimed to adequately model the relevant threat scenarios for CARBUROS.

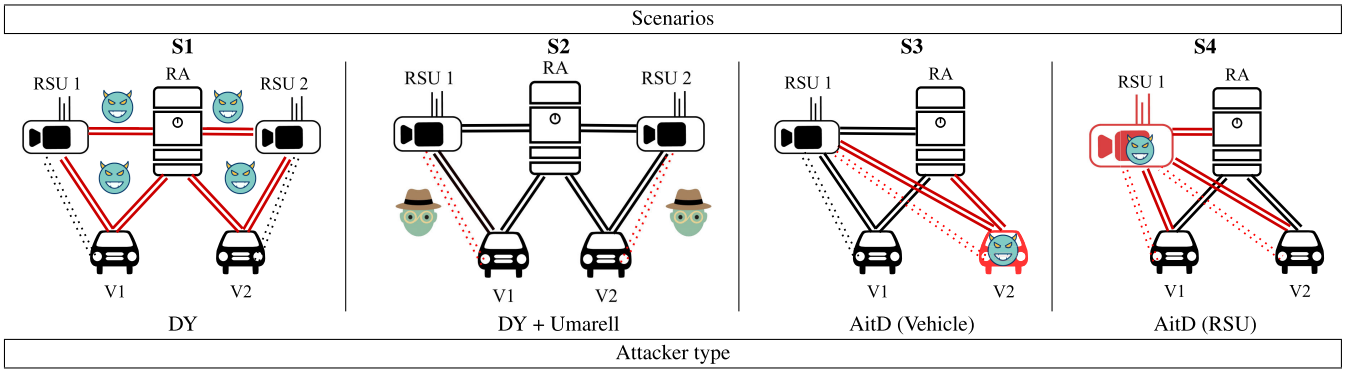
In particular, it is observed that:

1) the Umarell attacker alone cannot successfully hijack the protocol, eliminating the need to model such a scenario;
2) since identity swaps can only occur between two actors, there is no need to model scenarios involving more than two vehicles and RSUs;
3) in AitD scenarios, modeling a second RSU would be redundant as both vehicles have to interact with the single compromised endpoint that the attacker controls.

In the following, an overview of the ProVerif encoding for the four scenarios is provided, with particular emphasis on the distinguished feature described above.

The complete code can be found in the Appendix of this article.

TABLE 2. Description of the four scenarios: the man with the hat symbolizes the Umarell attacker, while the devil icon represents an attacker capable of controlling communication channels (highlighted in red) or an agent, as shown within the red agent icon, (AitD(X) denotes that X is under the control of AitD).



B. DATA ITEMS

CARBUROS relies on some data items briefly introduced in Section V. In the following, the implementation of these elements in ProVerif is detailed.

1) CRYPTOGRAPHIC PRIMITIVES

Following the standard ProVerif approach, the types for private and public keys, the primitives for public-key cryptographic operations (e.g., message encryption and signature), and the keys required for the protocol were introduced.

In particular, the following declarations were defined.

```
type skey. (* secret/private key type *)
type pkey. (* public key type *)

fun pk(skey): pkey. (* priv-pub key binding *)

(* encryption and decryption *)
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey;
             adec(aenc(m, pk(k)), k) = m.

(* sign, check and extract*)
fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, k: skey;
             checksign(sign(m, k), pk(k)) = m.
reduc forall m: bitstring, k: skey;
             getmess(sign(m, k)) = m.
```

Briefly, `skey` and `pkey` are the types for private and public keys, respectively. Each private key is uniquely associated with a public key through the `pk` function. Encryption is performed using the `aenc` function, which takes a message (recall that `bitstring` is the universal supertype in ProVerif) and a public key as inputs. Decryption is encoded as a reduction of the cyphertext `aenc(m, k)` to the cleartext `m`. The reduction, called `adec`, happens only when the proper private key is used, i.e., the one associated with the public key used to encrypt the message. Message signing is similar, but requires a private key. The signature can be checked while extracting the message from a digest using `checksign`. Finally, signed messages can be further inspected using the `getmess` function.

2) DATA AND DATA TABLES

Data items, such as *squawk* and *loc*, are treated as constants, i.e., immutable data items. All of them are of type `bitstring`. Moreover, since it is needed for the challenge-response interaction, the XOR operator $\oplus$ is modeled as follows.

```
(* XOR operator *)
fun xor(bitstring, bitstring): bitstring.
fun rox(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring;
                rox(m2, xor(m1, m2)) = m1.
```

Two functions are defined, i.e., `xor` and `rox`, and their relationship is stated in the `align` rule. This rule specifies that extracting the first element of a `xor` requires knowing the second element. Although both `xor` and `rox` represent the same operation, i.e., $\oplus$, they were distinguished to avoid circular dependencies that could result in undecidable models.

It is worth noticing that the modeling approach simplifies the actual behavior of $\oplus$, e.g., by not defining it symmetrically. However, this is irrelevant, as the attacker may only be interested in revealing the challenge in message 4 (see Figure 3). Furthermore, the bitmask does not play a role in the rest of the protocol.

The reversible function `rox` abstracts the bitwise behavior of the operation while maintaining logical equivalence. This modeling choice is consistent with standard symbolic analysis practices, as ProVerif focuses on message structure and derivability rather than low-level computation. For instance, associativity cannot be handled by ProVerif for this reason, which prevents the modeling of primitives such as XOR or groups [46].

C. CHANNELS

Communications relying on the network infrastructure are modeled using the standard mechanisms provided by ProVerif. In particular, variables of type `channel` are introduced as follows.

```
(* public channel *)
free V_RSU_chan: channel.
(* TLS-protected channel between vehicle and RA *)
free V_RA_chan: channel [private].
(* TLS-protected channel between RSU and RA *)
free RSU_RA_chan: channel [private].
```

- The channel V_RSU_chan is delivered by the RA at message 2 and it is unprotected.
- The other two channels V_RA_chan and RSU_RA_chan are, instead, private to the road infrastructure.

In scenarios where elements are duplicated, the corresponding channels were also replicated. For instance, when two RSUs were present, both RSU1_RA_chan and RSU2_RA_chan were defined.

1) PHYSICAL CHANNEL SIMULATION

Modeling the physical proximity-based channel requires special attention, as ProVerif lacks primitives to natively model proximity constraints or localized attackers, such as Umarell.

To address this issue, an intermediate agent, called PhyProxChan_Ag, was introduced to simulate both the behavior of the physical channel and the attacker interacting with it.

The implementation relies on four communication channels, each serving a distinct role in mediating and controlling message exchange.

- A private channel from the vehicle to the agent,
- A private channel from the agent to the RSU,
- A public channel for setting the operating mode, and
- A public channel for potential attacker access.

The code for the agent is as follows.

```
let PhyProxChan_Ag(from_V_chan: channel,
                   to_RSU_chan: channel,
                   mode_chan  : channel) =
  in (mode_chan, mode: bitstring);
  in (from_V_chan, msg: bitstring);

  if mode = eavesdrop then
    out (to_RSU_chan, msg);
    out (public_chan, msg).
```

This agent starts by reading the operating modality from the dedicated public channel (mode_chan). The modality is stored in the variable mode_chan, which is immutable once set.

It can take only one of two values: eavesdrop and destroy. By default, the eavesdrop modality is set before a vehicle uses the physical proximity-based channel, modeling the attacker's ability to observe messages. However, since mode_chan is public, the attacker can switch the mode to destroy, effectively preventing the message from reaching its destination. In both cases, the modality must be set before the actual transmission of the message and the attacker cannot change it after reading the message. Once the modality is set, the vehicle sends the message via a private channel. The agent then behaves as follows.

- If mode is set to eavesdrop, the message is sent to both the RSU (via the private channel to_RSU_chan) and the attacker (via the public channel public_chan).
- Otherwise, if mode is destroy, the message is not forwarded and therefore reaches neither the RSU nor the attacker.

D. AGENTS

Building on the elements discussed above, the vehicle agent is presented as an example of the agent modeling approach, while the details of the other two agents, the RSU and the RA, are provided in the Appendix.

1) VEHICLE AGENT

The Vehicle process is responsible for initiating communication with the RA, interacting with the RSU through both remote and physical proximity channels, and finally receiving the authentication token from the RA.

Technically, the ProVerif code of the vehicle process, reported in Listing 1, begins by defining the process as a function with several input parameters:

- *loc*: the vehicle's current location;
- v_RA_chan: the communication channel between the vehicle and the RA;
- agent_v_chan_proximity: the proximity channel, used later for the physical exchange;
- setting_mode_chan: a configuration or control channel;
- Prkey_vehicle: the vehicle's private signing key;
- squawk_Vehicle: the vehicle's *squawk* identifier;
- bitMask_Vehicle: a random mask used in challenge–response exchanges.

The steps are as follows.

Step 1 The vehicle V starts by sending its squawk and location to the RA (step 1 in the protocol, message (1) in the code), in the code with the primitive out. This initializes the session and tells the RA where the vehicle is currently located, allowing the RA to select the appropriate RSU for the next steps.

Step 2 Upon receiving the corresponding RSU data from the RA (step 2) (in the code with the primitive in), it identifies the correct RSU location and channel. This effectively assigns the vehicle to its nearest RSU for local communication.

Step 3 The vehicle then encrypts its squawk and bitmask with the RSU's public key and sends it (step 3). This establishes a secure digital channel to the RSU and prepares the next proximity-based phase. In the code, to this aim, *V* first looks up (getloctable) which RSU key (Prkey_RSU) corresponds to its location. It then selects the right RSU private key based on the RSU channel.

Step 4 After receiving the XOR of the challenge and the bitmask (the XOR is used to mask the challenge) from the RSU (step 4), the vehicle derives the original challenge, by using the shared bitMask_Vehicle and by reversing the XOR, with the function rox.

Step 5 Then the vehicle sends the challenge back over the physical proximity-based channel (step 5) (in [13] by flashing the required sequence). This is the proximity proof: it ensures that the vehicle is physically close enough to respond within the allowed time.

Step 6 Finally, the vehicle signs its squawk and the derived challenge and sends them to the RA (step 6). This step provides cryptographic proof of identity and linkage to the physical challenge received from the RSU.

Step 7 Step 7 does not involve V: it is carried out by the RSU, which sends a message containing a squawk and a challenge to the RA. The RA then compares this information with the data received directly from the vehicle V.

Step 8 If the check is successful, the RA generates a new authentication token, stores it, and sends it back to the vehicle (step 8), encrypted with the vehicle's public key, which can finally decrypt and verify it. This token is the final authentication proof, confirming that both digital and physical checks succeeded. More in detail, the vehicle decrypts it using its private key (`adec`), and verifies that it was correctly signed by the RA (`checksign`). The last output operation, the verified token is sent (through a given `oracle` channel) to an auxiliary process outside the protocol, called `Oracle`, used for verification in the ProVerif model.

E. SECURITY GOAL RESULTS

Finally, the standard query mechanism of ProVerif was employed to verify the goals defined in Section V-C, namely, ensuring that attackers cannot capture authentication tokens or compromise the authentication process, and to monitor the successful completion of authentication, as shown in Listing 2. All these queries are configured as reachability properties.

Referring to scenarios involving two vehicles (Table 2), the first two queries require that the secrecy of tokens *token1* and *token2* is preserved. Both tokens are bitstrings, initially declared as private and unknown to the attacker. Verification entails determining the set of terms (or messages) that the adversary could potentially learn during the protocol's execution. A message is deemed secret if it is absent from this set. This query is related to possible attacker objectives, such as *unauthorized access* and *identity swap*, as discussed in Subsection IV-A. In terms of *protocol failure*, the remaining query evaluates the reachability of the *success* event, which is emitted by a further auxiliary process outside the protocol, referred to as the Oracle. This event indicates whether the protocol has been completed successfully or unsuccessfully. The Oracle process verifies the correctness of the exchanged token by comparing it with a "magical" reference copy of the correct token. The event is considered reachable if the execution has been completed. ProVerif checks that the queries are valid for all protocol executions.

```
(* Vehicle process *)
let Vehicle(loc: bitstring, v_RA_chan: channel,
    agent_v_chan_proximity: channel,
    setting_mode_chan: channel, Prkey_vehicle:
    skey, squawk_Vehicle: bitstring,
    bitMask_Vehicle: bitstring) =
(* 1. V -> RA : squawk, loc *)
out (v_RA_chan, (squawk_Vehicle, loc));

(* 2. RA -> V : RSUdata *)
in (v_RA_chan, v_RSU_chan_vehicle: channel);

(* 3. V -> RSU : {squawk, bitmask}_pubRSU *)
get loctable(=loc, v_RSU_chan_vehicle) in
let Prkey_RSU = if v_RSU_chan_vehicle =
    v_RSU1_chan then Prkey_RSU1 else
    Prkey_RSU2 in
out (v_RSU_chan_vehicle, aenc((squawk_Vehicle,
     bitMask_Vehicle), pk(Prkey_RSU)));

(* 4. RSU -> V : xor(chall, bitmask) *)
out (setting_mode_channel, mode0);
in (v_RSU_chan_vehicle, chall_XOR_Vehicle:
    bitstring);

(* 5. V ->_phy RSU : chall *)
let chall_Vehicle = rox(bitMask_Vehicle,
    chall_XOR_Vehicle) in
out (agent_v_chan_proximity, chall_Vehicle);

(* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
    }_privV *)
out (v_RA_chan, sign((squawk_Vehicle,
    chall_Vehicle), Prkey_vehicle));

(* 8. RA -> V : {token}_privRA *)
in (v_RA_chan, token_signed_Vehicle: bitstring
    );
let token_digest_Vehicle = adec(
    token_signed_Vehicle, Prkey_vehicle) in
let token_Vehicle = checksign(
    token_digest_Vehicle, pk(Prkey_RA)) in
out (oracle_chan, (squawk_Vehicle,
    token_Vehicle)).
```

Listing 1. Vehicle process.

```
(* Queries *)
query attacker(token1).
query attacker(token2).
query event(success).
```

Listing 2. Security queries.

After running the ProVerif model for all four scenarios, the final results are those presented in Listing 3. The verification summary is as follows.

- *Query not attacker(token1[])* returned `true`: it means that the query is proved and that there is no attack. More in detail, ProVerif attempts to show *not attacker(token1)*, and hence *not attacker(token1)* being true means that the attacker did not gain access to `token1`. Therefore, its confidentiality is preserved.
- *Query not attacker(token2[])* returned `true`: similarly, the attacker did not gain access to `token2`, ensuring its confidentiality as well.
- *Query not event(success)* returned `false`: here ProVerif attempts to show that *not event(success)*. Obtaining `false` means that the event is reachable

and that the authentication process was successful as expected.

The attacker is unable to obtain either of the authentication tokens, and the process concludes successfully. This shows that CARBUROS is robust in each of the scenarios considered.

```
(* Verification summary *)
query not attacker(token1[]) is true.
query not attacker(token2[]) is true.
query not event(success) is false.
```

Listing 3. Verification results.

To conclude, the ProVerif analysis proved computationally lightweight. On an Intel Core i9-11900K processor, Scenario 1 completed in 67 ms, with all other scenarios consistently running between 60 ms and 70 ms. Incorporating extended attacker models (e.g., stateful compromises and additional message interception capabilities) did not significantly increase runtime, and ProVerif terminated without divergence or memory issues. These results indicate that the verification workload is modest and that the proposed models are practical for V2X protocol analysis. The most demanding part of the work lies upstream, in extending ProVerif to express new attacker models and scenarios, an effort required primarily at the beginning

VII. DISCUSSION

To further interpret these results, their implications and limitations were analyzed in the broader context of formal and practical validation. The identified challenges are discussed in three relevant aspects.

A. SYMBOLIC VERIFICATION

While exhaustive in its logical reasoning, symbolic verification abstracts away implementation-level aspects of security. Tools such as ProVerif assume perfect cryptographic primitives and cannot account for side-channel leakage, timing variations, or hardware-specific vulnerabilities. To bridge symbolic assurance with real-world protection, the mitigation of certain physical or insider threats through hardware-level safeguards was highlighted. For instance, the AitD scenario can be mitigated through hardware-anchored trust mechanisms. In practice, hardware roots of trust such as Trusted Platform Modules (TPMs) Hardware Security Modules (HSMs) can securely store long-term keys and enforce cryptographic operations, preventing key extraction or unauthorized software modification. Complementarily, remote attestation can verify the integrity and configuration of the device before initiating the CARBUROS protocol, ensuring that only uncompromised nodes participate in the authentication process. These countermeasures operate orthogonally to the symbolic verification presented in this work, strengthening resilience against insider threats while preserving the overall protocol logic. Future analyses could also examine additional properties, including liveness and fairness.

The described formal analysis guarantees that the CARBUROS protocol is sound under the symbolic DY model, Umarell and AitD, whereas its robustness against physical-layer interference, imperfect randomness, or real-world cryptographic attacks must be validated experimentally. These aspects are covered in the companion implementation study [13], which evaluates timing, interference resilience, and practical feasibility under realistic vehicular conditions.

To complement this discussion, the computational feasibility of the verification when introducing more complex adversary behaviors was also considered. Indeed, the introduction of additional attacker models (*Umarell* and AitD) expands the symbolic state space explored by ProVerif. Nevertheless, the verification remained tractable: all four modeled scenarios (DY, DY+Umarell, AitD–Vehicle, AitD–RSU) were analyzed successfully within a few minutes on a workstation equipped with a standard multi-core Central Processing Unit (CPU) and 16 GB of Random Access Memory (RAM). This result confirms that the extended attacker representation increases expressiveness without compromising scalability, making the approach practical.

B. QUANTITATIVE ANALYSIS

The verification results presented here are qualitative, as ProVerif operates under a symbolic model that establishes whether security properties hold or whether an attack exists, without quantifying the probability of success or the computational effort required by an adversary. This abstraction enables sound reasoning over an unbounded number of sessions but does not capture probabilistic or performance-related aspects such as resilience to message loss or resource-bounded attackers. Extending the analysis with quantitative assessments, through computational or simulation-based models, represents a valuable direction for future work to complement the symbolic guarantees provided by CARBUROS.

C. AI ATTACKS

Beyond classical symbolic and physical-layer threats, future extensions of the attacker model could consider AI-driven spoofing and adversarial machine learning attacks. These include manipulations targeting perception components such as camera- or sensor-based recognition systems, where crafted inputs could mislead classification or authentication mechanisms. Although such attacks fall outside the symbolic scope of CARBUROS, they are complementary to protocol-level verification. In this context, CARBUROS could interoperate with ML-based detection or sensor-fusion schemes to provide multi-layered protection combining perception robustness with formal proximity assurance.

VIII. CONCLUSION AND FUTURE WORK

Unlike conventional schemes that are based solely on digital credentials and PKIs, the proposed approach authenticates a vehicle not only through fixed credentials but also through contextual evidence, introducing a secondary

proximity-based verification factor that binds the authentication to the physical presence of the vehicle at a specific location and time. This concept is formalized in CARBUROS, a dual-channel authentication protocol that combines a standard digital communication channel with a physical proximity-based one. In this work, the remote channel enables efficient data exchange over conventional wireless networks, secured using TLS encryption as specified in IEEE 1609.2. The physical proximity-based channel, on the contrary, requires close-range interaction to confirm the physical presence of the communicating entity. In [13], this is obtained using vehicle headlights and RSU-mounted cameras.

CARBUROS was formally verified using ProVerif, demonstrating that, under defined assumptions and across several representative scenarios, it preserves key security properties, including the confidentiality of authentication tokens. These results validate the soundness of the design within the modeled conditions and provide confidence in its applicability to real vehicular environments. Furthermore, the formal verification confirms also the protocol's security properties under various attacker models, including both the classical DY adversary and the extended proximity-based attackers introduced in this work, validating its robustness against a broad spectrum of threats. The proposed layered approach of the protocol increases robustness against remote attacks while adding a physical dimension to authentication.

CARBUROS is designed to be modular, flexible, and channel-agnostic, which means that it operates independently of specific communication media and technologies. The protocol can accommodate a variety of physical channels, optical or based on alternative technologies, while preserving its core requirement: that the secondary channel enforces proximity by allowing message reception only within a constrained physical range. This flexibility allows for adjustments in message transmission and reception without requiring significant changes to the protocol itself. As long as the proximity constraint holds, the challenge-response logic and the dual-channel architecture remain unchanged. This modular design also facilitates the integration of newer technologies or adaptation to operational constraints (e.g., adverse lighting or weather conditions) while maintaining the security guaranties of the protocol.

A. FUTURE WORK: OTHER CHANNEL IMPLEMENTATIONS

Building on this channel-agnostic foundation, CARBUROS is planned to be adapted for use in RF-based systems, including 5G-V2X and future 6G communication frameworks, thereby extending its applicability to a wider range of secure V2I deployments, as well as the physical channel used to verify proximity, originally implemented using optical signals from vehicle headlights and RSU-mounted cameras, can be replaced with other technologies, such as photodetectors systems or mmWave radar units. These

```
set traceDisplay = long.
set attacker = active.

(* CARBUROS BASE AUTH PROTOCOL *)
(*
1. V -> RA      : squawk, loc
2. RA -> V      : RSUdata
3. V -> RSU     : {squawk, bitmask}_pubRSU
4. RSU -> V     : xor(chall, bitmask)
5. V ->_phy RSU: chall
6. V -> RA     : {squawk_Vehicle, chall_Vehicle}
    _privV
7. RSU -> RA    : squawk, chall
8. RA -> V     : {token}_privRA
*)

(* Types *)
type skey. (* private key *)
type pkey. (* public key *)

(* Channels *)
free v_RSU1_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v_RSU2_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v_RA_chan: channel [private]. (* private
    channel between vehicle and RA. TLS *)
free RSU_RA_chan : channel [private]. (* private
    channel between RSU and RA. TLS *)
free public_physical_chan: channel. (* public
    channel for agent mode 0 forwarding *)
free oracle_chan: channel [private]. (* private
    test channel *)

(* Private physical agent channels for
    communication between vehicle, agent, and RSU
    *)
free agent_v1_chan_physical: channel [private]. (*
     Private channel from vehicle to agent *)
free agent_RSU1_chan_physical: channel [private].
    (* Private channel from agent to RSU *)
free agent_v2_chan_physical: channel [private]. (*
     Private channel from vehicle to agent *)
free agent_RSU2_chan_physical: channel [private].
    (* Private channel from agent to RSU *)

(* Attacker-controllable channels for interacting
    with the physical channels *)
free setting_mode_chan_RSU1: channel.
free setting_mode_chan_RSU2: channel.

free token1: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk1: bitstring. (* squawk number,
    generated by RA and assigned during the
    enrollment *)
free bitMask1: bitstring [private]. (* Bit Mask,
    6-bit mask generated by the vehicle *)
free chall1: bitstring [private]. (* Challenge, 6-
    bit challenge generated by the RSU *)

free token2: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk2: bitstring. (* squawk number,
    generated by RA and assigned during the
    enrollment *)
free bitMask2: bitstring [private]. (* Bit Mask,
    6-bit mask generated by the vehicle *)
free chall2: bitstring [private]. (* Challenge, 6-
    bit challenge generated by the RSU *)
```

Listing 4. ProVerif code for Scenario 1-2.

```
(* Keys *)
free Prkey_vehicle1: skey [private].
free Prkey_vehicle2: skey [private].
free Prkey_RA: skey [private].
free Prkey_RSU1: skey [private].
free Prkey_RSU2: skey [private].
(* Mode constants *)
free eavesdrop: bitstring.
free destroy: bitstring.

(* Tables *)
table keytable(bitstring, pkey). (* maps squawk to
     keys *)
table toktable(bitstring, bitstring). (* maps
    squawk to the right token *)
table loctable(bitstring, channel). (* maps
    location to RSU *)

(* Constructors *)

(* Cryptographic functions *)
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m,
    pk(k)), k) = m.

(* fun hash(bitstring): bitstring. *)

fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, k: skey; checksign(sign
    (m, k), pk(k)) = m.
reduc forall m: bitstring, k: skey; getmess(sign(m
    , k)) = m.

(* XOR *)
fun xor(bitstring, bitstring): bitstring.
fun rox(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring; rox(
    m2, xor(m1, m2)) = m1.

(* Events *)
event success.

(* Queries *)
query attacker(token1).
query attacker(token2).
query event(success).

(* Processes Auth *)

(* Vehicle process *)
let Vehicle(loc: bitstring, v_RA_chan: channel,
    agent_v_chan_physical: channel,
    setting_mode_chan: channel, Prkey_vehicle:
    skey, squawk_Vehicle: bitstring,
    bitMask_Vehicle: bitstring) =

    (* 1. V -> RA : squawk, loc *)
    out (v_RA_chan, (squawk_Vehicle, loc));

    (* 2. RA -> V : RSUdata *)
    in (v_RA_chan, v_RSU_chan_vehicle: channel);

    (* 3. V -> RSU : {squawk, bitmask}_pubRSU *)
    get loctable(=loc, v_RSU_chan_vehicle) in (*
        Look up the corresponding RSU for the
        vehicle's location *)
    let Prkey_RSU = if v_RSU_chan_vehicle =
        v_RSU1_chan then Prkey_RSU1 else
        Prkey_RSU2 in (* Fetch the public key of
        the correct RSU based on location *)
    out (v_RSU_chan_vehicle, aenc((squawk_Vehicle,
         bitMask_Vehicle), pk(Prkey_RSU)));
```

Listing 4. *(Continued.)* ProVerif code for Scenario 1-2.

```
    (* 4. RSU -> V : xor(chall, bitmask) *)
    out (setting_mode_chan, eavesdrop); (*
        Necessary to enable the physical chan *)
    in (v_RSU_chan_vehicle, chall_XOR_Vehicle:
        bitstring);
    (* 5. V ->_phy RSU : chall *)
    let chall_Vehicle = rox(bitMask_Vehicle,
        chall_XOR_Vehicle) in
    out (agent_v_chan_physical, chall_Vehicle); (*
         Send challenge to the physical agent *)

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    out (v_RA_chan, sign((squawk_Vehicle,
        chall_Vehicle), Prkey_vehicle));

    (* 8. RA -> V : {token}_privRA *)
    in (v_RA_chan, token_signed_Vehicle: bitstring
        );
    let token_digest_Vehicle = adec(
        token_signed_Vehicle, Prkey_vehicle) in
    let token_Vehicle = checksign(
        token_digest_Vehicle, pk(Prkey_RA)) in
    out (oracle_chan, (squawk_Vehicle,
        token_Vehicle)).


(* RSU process *)
let RSU(agent_RSU_chan_physical: channel,
    v_RSU_chan: channel, chall_RSU: bitstring,
    Prkey_RSU: skey) =

    (* 3. V -> RSU    : {squawk, bitmask}_pubRSU
        *)
    in (v_RSU_chan, chall_info: bitstring);

    (* 4. RSU -> V    : xor(chall, bitmask) *)
    let (squawk_RSU: bitstring, bitMask_RSU:
        bitstring) = adec(chall_info, Prkey_RSU)
        in
    out (v_RSU_chan, xor(chall_RSU, bitMask_RSU));

    (* 5. V ->_phy RSU: chall *)
    in (agent_RSU_chan_physical, =chall_RSU); (*
        Receive challenge from the physical agent
        *)

    (* 7. RSU -> RA   : squawk, chall *)
    out (RSU_RA_chan , (squawk_RSU, chall_RSU)).

(* RA process *)
let RA() =

    (* 1. V -> RA : squawk, loc *)
    in (v_RA_chan, (squawk_RA: bitstring, loc:
        bitstring));

    (* 2. RA -> V : RSUdata *)
    get loctable(=loc, v_RSU_RA) in
    get keytable(=squawk_RA, PubKey_vehicle) in
    out (v_RA_chan, v_RSU_RA);

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    in (v_RA_chan, signed_msg_RA: bitstring);

    (* 7. RSU -> RA   : squawk, chall *)
    in (RSU_RA_chan , (=squawk_RA, RA_C_t:
        bitstring));
```

Listing 4. *(Continued.)* ProVerif code for Scenario 1-2.

```
            (* 8. RA -> V : {token}_privRA *)
            out (v_RA_chan, aenc(sign(token_RA,
                Prkey_RA), PubKey_vehicle)).


(* ORACLE process *)
let ORACLE =
    in (oracle_chan, (squawk_ORACLE: bitstring,
        token_ORACLE: bitstring));
    get toktable(=squawk_ORACLE, right_token) in
    if token_ORACLE = right_token
        then event success.

(* physical agent process *)
let PhyProxChan_Ag(agent_v_chan_physical: channel,
     agent_RSU_chan_physical: channel,
    setting_mode_chan: channel) =

    in (setting_mode_chan, mode: bitstring);

    in (agent_v_chan_physical, msg: bitstring);

    (* Pattern matching on mode *)
    (* If mode is eavesdrop, forward the message
        to both RSU and public chan *)
    if mode = eavesdrop then
        out (agent_RSU_chan_physical, msg); (*
            Forward to RSU *)
        out (public_physical_chan, msg). (*
            Forward to public chan *)

(* Final process block *)
process
    new loc1: bitstring;
    new loc2: bitstring;
    insert keytable(squawk1, pk(Prkey_vehicle1));
    insert keytable(squawk2, pk(Prkey_vehicle2));
    insert loctable(loc1, v_RSU1_chan);
    insert loctable(loc2, v_RSU2_chan);

    (* Vehicle 1 and physical Agent 1 for RSU1 *)
    Vehicle(loc1, v_RA_chan,
        agent_v1_chan_physical,
        setting_mode_chan_RSU1, Prkey_vehicle1,
        squawk1, bitMask1) |
    PhyProxChan_Ag(agent_v1_chan_physical,
        agent_RSU1_chan_physical,
        setting_mode_chan_RSU1) |

    (* Vehicle 2 and physical Agent 2 for RSU2 *)
    Vehicle(loc2, v_RA_chan,
        agent_v2_chan_physical,
        setting_mode_chan_RSU2, Prkey_vehicle2,
        squawk2, bitMask2) |
    PhyProxChan_Ag(agent_v2_chan_physical,
        agent_RSU2_chan_physical,
        setting_mode_chan_RSU2) |

    (* RSUs and RA *)
    RSU(agent_RSU1_chan_physical, v_RSU1_chan,
        chall1, Prkey_RSU1) |
    RSU(agent_RSU2_chan_physical, v_RSU2_chan,
        chall2, Prkey_RSU2) |
    RA() | ORACLE()
```

Listing 4. ***(Continued.)*** **ProVerif code for Scenario 1-2.**

technologies introduce features such as Ultra-Reliable Low-Latency Communication (URLLC), network slicing, and edge computing, which can further enhance the efficiency and

```
set traceDisplay = long.
set attacker = active.

(* Types *)
type skey. (* private key *)
type pkey. (* public key *)
(* Channels *)
free v1_RSU1_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v2_RSU1_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v1_RA_chan: channel [private]. (* private
    channel between vehicle1 and RA. TLS *)
free v2_RA_chan: channel . (* public channel,
    Dolev-Yao model Open Channel between vehicle2
    and RA.*)
free v1_RSU1_chan_physical: channel. (* public
    physical channel from vehicle1 to RSU *)
free v2_RSU1_chan_physical: channel. (* public
    physical channel from vehicle2 to RSU *)
free RSU_RA_chan: channel [private]. (* private
    channel between RSU and RA. TLS *)
free oracle_chan: channel [private]. (* private
    test channel *)

free token1: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk1: bitstring [private]. (* squawk
    number, generated by RA and assigned during
    the enrollement*)
free bitMask1: bitstring [private]. (* Bit Mask,
    6-bit mask generated by the vehicle *)
free chall1: bitstring [private]. (* Challenge, 6-
    bit challenge generated by the RSU *)

free token2: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk2: bitstring. (* squawk number,
    generated by RA and assigned during the
    enrollement*)
free bitMask2: bitstring. (* Bit Mask, 6-bit mask
    generated by the vehicle *)
free chall2: bitstring. (* Challenge, 6-bit
    challenge generated by the RSU *)

(* Keys *)
free Prkey_vehicle1: skey [private].
free Prkey_vehicle2: skey.
free Prkey_RA: skey [private].
free Prkey_RSU1: skey [private].

(* Tables *)
table keytable(bitstring, pkey). (* maps squawk to
     keys *)
table toktable(bitstring, bitstring). (* maps
    squawk to the right token *)

(* Constructors *)

(* Cryptographic functions *)
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m,
    pk(k)), k) = m.

(* fun hash(bitstring): bitstring. *)

fun sign(bitstring, skey): bitstring.
```

Listing 5. **ProVerif code for Scenario 3.**

```
reduc forall m: bitstring, k: skey; checksign(sign
    (m, k), pk(k)) = m.
reduc forall m: bitstring, k: skey; getmess(sign(m
    , k)) = m.
(* XOR *)
fun xor(bitstring, bitstring): bitstring.
fun rox(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring; rox(
    m2, xor(m1, m2)) = m1.

(* Events *)
event success.

(* Queries *)
query attacker(token1).
query attacker(token2).
query event(success).

(* Processes Auth *)
(* Vehicle process *)
let Vehicle(loc: bitstring, v_RA_chan: channel,
    v_RSU_chan_physical: channel, Prkey_vehicle:
    skey, squawk_Vehicle: bitstring,
    bitMask_Vehicle: bitstring) =

    (* 1. V -> RA : squawk, loc *)
    out (v_RA_chan, (squawk_Vehicle, loc));

    (* 2. RA -> V : RSUdata *)
    in (v_RA_chan, v_RSU_chan_vehicle: channel);

    (* 3. V -> RSU : {squawk, bitmask}_pubRSU *)
    let Prkey_RSU = Prkey_RSU1 in (* Retrieve the
        public key of the RSU *)
    out (v_RSU_chan_vehicle, aenc((squawk_Vehicle,
         bitMask_Vehicle), pk(Prkey_RSU)));

    (* 4. RSU -> V : xor(chall, bitmask) *)
    in (v_RSU_chan_vehicle, chall_XOR_Vehicle:
        bitstring);

    (* 5. V ->_phy RSU : chall *)
    let chall_Vehicle = rox(bitMask_Vehicle,
        chall_XOR_Vehicle) in
    out (v_RSU_chan_physical, chall_Vehicle);

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    out (v_RA_chan, sign((squawk_Vehicle,
        chall_Vehicle), Prkey_vehicle));

    (* 8. RA -> V : {token}_privRA *)
    in (v_RA_chan, token_signed_Vehicle: bitstring
        );
    let token_digest_Vehicle = adec(
        token_signed_Vehicle, Prkey_vehicle) in
    let token_Vehicle = checksign(
        token_digest_Vehicle, pk(Prkey_RA)) in
    out (oracle_chan, (squawk_Vehicle,
        token_Vehicle)).

(* RSU process *)
let RSU(v_RSU_chan: channel, v_RSU_chan_physical:
    channel, chall_RSU: bitstring, Prkey_RSU: skey
    ) =
    (* 3. V -> RSU    : {squawk, bitmask}_pubRSU
        *)
    in (v_RSU_chan, chall_info: bitstring);

    (* 4. RSU -> V    : xor(chall, bitmask) *)
    let (squawk_RSU: bitstring, bitMask_RSU:
        bitstring) = adec(chall_info, Prkey_RSU)
        in
    out (v_RSU_chan, xor(chall_RSU, bitMask_RSU));
```

Listing 5. *(Continued.)* ProVerif code for Scenario 3.

```
    (* 5. V ->_phy RSU: chall *)
    in (v_RSU_chan_physical, =chall_RSU); (*
        Receive challenge from the physical agent
        *)

    (* 7. RSU -> RA   : squawk, chall *)
    out (RSU_RA_chan, (squawk_RSU, chall_RSU)).

(* RA process *)
let RA(v_RA_chan: channel, v_RSU_chan:channel) =

    (* 1. V -> RA : squawk, loc *)
    in (v_RA_chan, (squawk_RA: bitstring, loc:
        bitstring));

    (* 2. RA -> V : RSUdata *)
    get keytable(=squawk_RA, PubKey_vehicle) in
    out (v_RA_chan, v_RSU_chan);

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    in (v_RA_chan, signed_msg_RA: bitstring);

    (* 7. RSU -> RA   : squawk, chall *)
    in (RSU_RA_chan, (=squawk_RA, RA_C_t:
        bitstring));

    let (squawk_RA_recv: bitstring, chall_RA_recv:
         bitstring) = checksign(signed_msg_RA,
        PubKey_vehicle) in
    if (squawk_RA_recv = squawk_RA) then
        (* Check *)
        if (chall_RA_recv = RA_C_t) then
            new token_RA: bitstring;
            insert toktable(squawk_RA, token_RA);

            (* 8. RA -> V : {token}_privRA *)
            out (v_RA_chan, aenc(sign(token_RA,
                Prkey_RA), PubKey_vehicle)).

(* ORACLE process *)
let ORACLE =
    in (oracle_chan, (squawk_ORACLE: bitstring,
        token_ORACLE: bitstring));
    get toktable(=squawk_ORACLE, right_token) in
    if token_ORACLE = right_token
        then event success.

(* Final process block *)
process
    new loc1: bitstring;
    insert keytable(squawk1, pk(Prkey_vehicle1));
    insert keytable(squawk2, pk(Prkey_vehicle2));

    Vehicle(loc1, v1_RA_chan,
        v1_RSU1_chan_physical, Prkey_vehicle1,
        squawk1, bitMask1) |
    Vehicle(loc1, v2_RA_chan,
        v2_RSU1_chan_physical, Prkey_vehicle2,
        squawk2, bitMask2) |

    RSU(v1_RSU1_chan, v1_RSU1_chan_physical,
        chall1, Prkey_RSU1) |
    RSU(v2_RSU1_chan, v2_RSU1_chan_physical,
        chall2, Prkey_RSU1) |
    RA(v1_RA_chan, v1_RSU1_chan) |
    RA(v2_RA_chan, v2_RSU1_chan) |
    ORACLE()
```

Listing 5. *(Continued.)* ProVerif code for Scenario 3.

```
set traceDisplay = long.
set attacker = active.

(* Types *)
type skey. (* private key *)
type pkey. (* public key *)

(* Channels *)
free v1_RSU1_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v2_RSU1_chan: channel. (* public channel,
    Dolev-Yao model Open Channel *)
free v1_RA_chan: channel [private]. (* private
    channel between vehicle1 and RA. TLS *)
free v2_RA_chan: channel . (* private channel
    between vehicle2 and RA. TLS *)
free v1_RSU1_chan_physical: channel. (* public
    physical channel from vehicle1 to RSU *)
free v2_RSU1_chan_physical: channel. (* public
    physical channel from vehicle2 to RSU *)
free RSU_RA_chan: channel . (* private channel
    between RSU and RA. TLS *)
free oracle_chan: channel [private]. (* private
    test channel *)

free token1: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk1: bitstring . (* squawk number,
    generated by RA and assigned during the
    enrollement*)
free bitMask1: bitstring . (* Bit Mask, 6-bit mask
     generated by the vehicle *)
free chall1: bitstring . (* Challenge, 6-bit
    challenge generated by the RSU *)

free token2: bitstring [private]. (* token,
    generated by RA, to authenticate the vehicle
    *)
free squawk2: bitstring . (* squawk number,
    generated by RA and assigned during the
    enrollement*)
free bitMask2: bitstring. (* Bit Mask, 6-bit mask
    generated by the vehicle *)
free chall2: bitstring. (* Challenge, 6-bit
    challenge generated by the RSU *)

(* Keys *)
free Prkey_vehicle1: skey [private].
free Prkey_vehicle2: skey [private].
free Prkey_RA: skey [private].
free Prkey_RSU1: skey.

(* Tables *)
table keytable(bitstring, pkey). (* maps squawk to
     keys *)
table toktable(bitstring, bitstring). (* maps
    squawk to the right token *)

(* Constructors *)

(* Cryptographic functions *)
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m,
    pk(k)), k) = m.

(* fun hash(bitstring): bitstring. *)

fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, k: skey; checksign(sign
    (m, k), pk(k)) = m.
reduc forall m: bitstring, k: skey; getmess(sign(m
    , k)) = m.
```

Listing 6. ProVerif code for Scenario 4.

```
(* XOR *)
fun xor(bitstring, bitstring): bitstring.
fun rox(bitstring, bitstring): bitstring.
equation forall m1: bitstring, m2: bitstring; rox(
    m2, xor(m1, m2)) = m1.

(* Events *)
event success.

(* Queries *)
query attacker(token1).
query attacker(token2).
query event(success).

(* Processes Auth *)
(* Vehicle process *)
let Vehicle(loc: bitstring, v_RA_chan: channel,
    v_RSU_chan_physical: channel, Prkey_vehicle:
    skey, squawk_Vehicle: bitstring,
    bitMask_Vehicle: bitstring) =

    (* 1. V -> RA : squawk, loc *)
    out (v_RA_chan, (squawk_Vehicle, loc));

    (* 2. RA -> V : RSUdata *)
    in (v_RA_chan, v_RSU_chan_vehicle: channel);

    (* 3. V -> RSU : {squawk, bitmask}_pubRSU *)
    let Prkey_RSU = Prkey_RSU1 in (* Retrieve the
        public key of the RSU *)
    out (v_RSU_chan_vehicle, aenc((squawk_Vehicle,
         bitMask_Vehicle), pk(Prkey_RSU)));

    (* 4. RSU -> V : xor(chall, bitmask) *)
    in (v_RSU_chan_vehicle, chall_XOR_Vehicle:
        bitstring);

    (* 5. V ->_phy RSU : chall *)
    let chall_Vehicle = rox(bitMask_Vehicle,
        chall_XOR_Vehicle) in
    out (v_RSU_chan_physical, chall_Vehicle);

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    out (v_RA_chan, sign((squawk_Vehicle,
        chall_Vehicle), Prkey_vehicle));

    (* 8. RA -> V : {token}_privRA *)
    in (v_RA_chan, token_signed_Vehicle: bitstring
        );
    let token_digest_Vehicle = adec(
        token_signed_Vehicle, Prkey_vehicle) in
    let token_Vehicle = checksign(
        token_digest_Vehicle, pk(Prkey_RA)) in
    out (oracle_chan, (squawk_Vehicle,
        token_Vehicle)).

(* RSU process *)
let RSU(v_RSU_chan: channel, v_RSU_chan_physical:
    channel, chall_RSU: bitstring, Prkey_RSU: skey
    ) =

    (* 3. V -> RSU    : {squawk, bitmask}_pubRSU
        *)
    in (v_RSU_chan, chall_info: bitstring);

    (* 4. RSU -> V    : xor(chall, bitmask) *)
    let (squawk_RSU: bitstring, bitMask_RSU:
        bitstring) = adec(chall_info, Prkey_RSU)
        in
    out (v_RSU_chan, xor(chall_RSU, bitMask_RSU));
```

Listing 6. *(Continued.)* ProVerif code for Scenario 4.

```
    (* 5. V ->_phy RSU: chall *)
    in (v_RSU_chan_physical, =chall_RSU); (*
        Receive challenge from the physical agent
        *)

    (* 7. RSU -> RA    : squawk, chall *)
    out (RSU_RA_chan, (squawk_RSU, chall_RSU)).

(* RA process *)
let RA(v_RA_chan: channel, v_RSU_chan:channel) =

    (* 1. V -> RA : squawk, loc *)
    in (v_RA_chan, (squawk_RA: bitstring, loc:
        bitstring));

    (* 2. RA -> V : RSUdata *)
    get keytable(=squawk_RA, PubKey_vehicle) in
    out (v_RA_chan, v_RSU_chan);

    (* 6. V -> RA : {squawk_Vehicle, chall_Vehicle
        }_privV *)
    in (v_RA_chan, signed_msg_RA: bitstring);

    (* 7. RSU -> RA    : squawk, chall *)
    in (RSU_RA_chan, (=squawk_RA, RA_C_t:
        bitstring));

    let (squawk_RA_recv: bitstring, chall_RA_recv:
         bitstring) = checksign(signed_msg_RA,
        PubKey_vehicle) in
    if (squawk_RA_recv = squawk_RA) then
        (* Check *)
        if (chall_RA_recv = RA_C_t) then
            new token_RA: bitstring;
            insert toktable(squawk_RA, token_RA);

            (* 8. RA -> V : {token}_privRA *)
            out (v_RA_chan, aenc(sign(token_RA,
                Prkey_RA), PubKey_vehicle)).

(* ORACLE process *)
let ORACLE =
    in (oracle_chan, (squawk_ORACLE: bitstring,
        token_ORACLE: bitstring));
    get toktable(=squawk_ORACLE, right_token) in
    if token_ORACLE = right_token
        then event success.

(* Final process block *)
process
    new loc1: bitstring;
    insert keytable(squawk1, pk(Prkey_vehicle1));
    insert keytable(squawk2, pk(Prkey_vehicle2));

    Vehicle(loc1, v1_RA_chan,
        v1_RSU1_chan_physical, Prkey_vehicle1,
        squawk1, bitMask1) |
    Vehicle(loc1, v2_RA_chan,
        v2_RSU1_chan_physical, Prkey_vehicle2,
        squawk2, bitMask2) |

    RSU(v1_RSU1_chan, v1_RSU1_chan_physical,
        chall1, Prkey_RSU1) |
    RSU(v2_RSU1_chan, v2_RSU1_chan_physical,
        chall2, Prkey_RSU1) |
    RA(v1_RA_chan, v1_RSU1_chan) |
    RA(v2_RA_chan, v2_RSU1_chan) |
    ORACLE()
```

Listing 6. ***(Continued.)*** **ProVerif code for Scenario 4.**

scalability of the dual-channel authentication process. Using the distributed edge infrastructure of 5G, CARBUROS could enable faster challenge response validation and improved proximity verification in dense or dynamic environments.

B. FUTURE WORK: OTHER APPLICATION SCENARIOS

Beyond the automotive domain, CARBUROS can be adapted to other transportation sectors where verifying physical proximity is crucial and fixed infrastructure may be limited, such as maritime and aviation with Unmanned Aerial Vehicles (UAVs). Its core concept, which combines cryptographic communication with physical presence verification, offers a generalizable and future-oriented approach to secure authentication. Future work will explore the extension of CARBUROS to multi-vehicle interaction scenarios and its integration across different transportation modes, reinforcing its potential as a scalable, secure, and adaptable authentication protocol for ITSs and beyond.

APPENDIX

A. SCENARIO 1-2

See Listing 4.

B. SCENARIO 3

See Listing 5.

C. SCENARIO 4

See Listing 6.

ACKNOWLEDGMENT

The authors would like to thank Prof. Sanjay Sarma and Prof. Dajiang Suo for their insightful suggestions, including the concept of dual-channel communication.

REFERENCES

[1] C. Bodei, M. D. Vincenzi, and I. Matteucci, "From hardware-functional to software-defined vehicles and their security issues," in *Proc. IEEE 21st Int. Conf. Ind. Informat. (INDIN)*, Jul. 2023, pp. 1–10, doi: 10.1109/INDIN51400.2023.10217971.

[2] *IEEE Standard for Wireless Access in Vehicular Environments-Security Services for Application and Management Messages*, Standard 1609.2-2022, 2023.

[3] M. De Vincenzi, J. Moore, B. Smith, S. E. Sarma, and I. Matteucci, "Security risks and designs in the connected vehicle ecosystem: In-vehicle and edge platforms," *IEEE Open J. Veh. Technol.*, vol. 6, pp. 442–454, 2025.

[4] M. De Vincenzi, M. D. Pesé, C. Bodei, I. Matteucci, R. R. Brooks, M. Hasan, A. Saracino, M. Hamad, and S. Steinhorst, "Contextualizing security and privacy of software-defined vehicles: State of the art and industry perspectives," 2024, *arXiv:2411.10612*.

[5] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE)—Certificate Management Interfaces for End Entities*, Standard 1609.2.1-2022, 2022.

[6] *Intelligent Transport Systems—ITS Station Security Services for Secure Session Establishment and Authentication Between Trusted Devices*, Standard 177:2024, 2024.

[7] M. Wolf and D. Serpanos, "Safety and security in cyber-physical systems and Internet-of-Things systems," *Proc. IEEE*, vol. 106, no. 1, pp. 9–20, Jan. 2018.

[8] X. Sun, F. R. Yu, and P. Zhang, "A survey on cyber-security of connected and autonomous vehicles (CAVs)," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6240–6259, Jul. 2022.

[9] G. Costantino, M. De Vincenzi, and I. Matteucci, "In-depth exploration of ISO/SAE 21434 and its correlations with existing standards," *IEEE Commun. Standards Mag.*, vol. 6, no. 1, pp. 84–92, Mar. 2022.

[10] (2020). *Trier: Five Die As Car Ploughs Through Germany Pedestrian Zone*. Accessed: Oct. 24, 2025. [Online]. Available: https://www.bbc.com/news/world-europe-55148518

[11] U.S. Department of Transportation, Intelligent Transportation Systems. *Intelligent Transportation Systems Joint Program Office*. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/43635

[12] (2021). *An Overview of Harmonised Cooperative Intelligent Transport Systems (C-ITS) Deployment in Europe*. [Online]. Available: https://www.c-roads.eu/fileadmin/userupload/media/Dokumente/C-Road sBrochure2021final2.pdf

[13] M. D. Vincenzi, S. Sun, C. B. C. Zhang, M. Garcia, S. Ding, C. Bodei, I. Matteucci, S. E. Sarma, and D. Suo, "Vehicular communication security: Multi-channel and multi-factor authentication," *IEEE Trans. Veh. Technol.*, early access, Aug. 12, 2025, doi: 10.1109/TVT.2025.3598113.

[14] B. Dwyer, S. E. Sarma, and D. Suo, "Enabling secure vehicle to infrastructure communication via two-factor authentication," in *Proc. IEEE 26th Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2023, pp. 5663–5668, doi: 10.1109/ITSC57777.2023.10421946.

[15] D. Suo and S. E. Sarma, "A two-factor authentication scheme for moving connected vehicles," in *Proc. IEEE 96th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2022, pp. 1–5, doi: 10.1109/VTC2022-FALL57202.2022.10012773.

[16] B. Blanchet and V. Cheval. (2021). *ProVerif: Cryptographic Protocol Verifier in the Formal Model*. Accessed: Oct. 24, 2025. [Online]. Available: https://bblanche.gitlabpages.inria.fr/proverif

[17] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983, doi: 10.1109/TIT.1983.1056650.

[18] A. Alsoliman, M. Levorato, and Q. A. Chen, "Vision-based two-factor authentication & localization scheme for autonomous vehicles," in *Proc. 3rd Int. Workshop Automot. Auto. Vehicle Secur.*, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/autosec202123021paper.pdf

[19] S. Meier-Vieracker, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Proc. Comput. Aided Verification (CAV)*, 2013, pp. 696–701.

[20] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. v. Oheimb, M. Rusinowitch, J. S. Santiago, M. Turuani, L. Viganó, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Proc. Comput. Aided Verification (CAV)*, 2005, pp. 281–285.

[21] C. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols," in *Proc. Comput. Aided Verification (CAV)*, Mali, 2008, pp. 414–418.

[22] Y. Belfaik, Y. Lotfi, Y. Sadqi, and S. Safi, "A comparative study of protocols' security verification tools: Avispa, scyther, proverif, and tamarin," in *Digital Technologies and Applications*, S. Motahhir and B. Bossoufi, Eds., Cham, Switzerland: Springer, 2024, pp. 118–128.

[23] A. Bruni, M. Sójka, F. Nielson, and H. R. Nielson, "Formal security analysis of the MaCAN protocol," in *Proc. Int. Conf. Integr. Formal Methods*, Bertinoro, Italy, 2014, pp. 241–255.

[24] T. Lauser, D. Zelle, and C. Krauß, "Security analysis of automotive protocols," in *Proc. Comput. Sci. Cars Symp.*, Dec. 2020, pp. 1–12, doi: 10.1145/3385958.3430482.

[25] M. Krichen, "Formal methods and validation techniques for ensuring automotive systems security," *Information*, vol. 14, no. 12, p. 666, Dec. 2023, doi: 10.3390/info14120666. [Online]. Available: https://www.mdpi.com/2078-2489/14/12/666

[26] C. Bodei, M. De Vincenzi, and I. Matteucci, "Formal analysis of an AUTOSAR-based basic software module," *Int. J. Softw. Tools for Technol. Transf.*, vol. 26, no. 4, pp. 495–508, Aug. 2024, doi: 10.1007/s10009-024-00759-w.

[27] G. Bella, P. Biondi, G. Costantino, and I. Matteucci, "Designing and implementing an AUTOSAR-based basic software module for enhanced security," *Comput. Netw.*, vol. 218, Dec. 2022, Art. no. 109377, doi: 10.1016/j.comnet.2022.109377.

[28] G. Bella, P. Biondi, G. Costantino, and I. Matteucci, "CINNAMON: A module for AUTOSAR secure onboard communication," in *Proc. 16th Eur. Dependable Comput. Conf. (EDCC)*, Sep. 2020, pp. 103–110.

[29] G. Costa, P. Degano, L. Galletta, and S. Soderi, "Formally verifying security protocols built on watermarking and jamming," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103133, doi: 10.1016/j.cose.2023.103133.

[30] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," in *Proc. IEEE 31st Comput. Secur. Found. Symp. (CSF)*, Jul. 2018, pp. 1–15, doi: 10.1109/CSF.2018.00008.

[31] P. Van Aubel and E. Poll, "Security of EV-charging protocols," 2022, *arXiv:2202.04631*.

[32] S. Mookherji, V. Odelu, and R. Prasath, "Secure ultra fast authentication protocol for electric vehicle charging," *Comput. Electr. Eng.*, vol. 119, Oct. 2024, Art. no. 109512.

[33] S. Hamdare, D. J. Brown, D. N. Jha, M. Aljaidi, Y. Cao, S. Kumar, R. Kharel, M. Jugran, and O. Kaiwartya, "Cyber defense in OCPP for EV charging security risks," *Int. J. Inf. Secur.*, vol. 24, no. 3, Jun. 2025.

[34] Z. Ying, K. Wang, J. Xiong, and M. Ma, "A literature review on V2Xcommunications security: Foundation, solutions, status, and future," *IET Commun.*, vol. 18, pp. 1683–1715, 2024, doi: 10.1049/cmu2.12778.

[35] H. Tanyıldız, C. B. Şahin, Ö. B. Dinler, H. Migdady, K. Saleem, A. Smerat, A. H. Gandomi, and L. Abualigah, "Detection of cyber attacks in electric vehicle charging systems using aremaining useful life generative adversarial network," *Sci. Rep.*, vol. 15, 2025, doi: 10.1038/s41598-025-92895-9.

[36] I. Qiqieh, O. Alzubi, J. Alzubi, K. C. Sreedhar, and A. M. Al-Zoubi, "An intelligent cyber threat detection: A swarm-optimized machine learning approach," *Alexandria Eng. J.*, vol. 115, pp. 553–563, Mar. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110016824016351

[37] O. A. Alzubi, J. A. Alzubi, M. Alazab, A. Alrabea, A. Awajan, and I. Qiqieh, "Optimized machine learning-based intrusion detection system for fog and edge computing environment," *Electronics*, vol. 11, no. 19, p. 3007, Sep. 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/19/3007

[38] S. H. V. Bhupathiraju, T. Sato, M. Clifford, T. Sugawara, Q. A. Chen, and S. Rampazzi, "On the vulnerability of traffic light recognition systems to laser illumination attacks," in *Proc. Symp. Vehicle Secur. Privacy*, 2024. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/on-the-vulnerability-of-traffic-light-recognition-systems-to-laser-illumination-attacks/

[39] F. Sinigaglia, R. Carbone, G. Costa, and N. Zannone, "A survey on multi-factor authentication for online banking in the wild," *Comput. Secur.*, vol. 95, Aug. 2020, Art. no. 101745. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820300316

[40] T. Dougan and K. Curran, "Man in the browser attacks," *Int. J. Ambient Comput. Intell.*, vol. 4, no. 1, pp. 29–39, Jan. 2012.

[41] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2027–2051, 3rd Quart., 2016.

[42] *Intelligent Transport Systems—ITS Station Security Services for Secure Session Establishment and Authentication Between Trusted Devices*, Standard 22 177:2024, 2024, p. 2024. [Online]. Available: https://www.iso.org/standard/87225.html

[43] (2020). *What Are Squawk Codes?*. Accessed: Oct. 24, 2025. [Online]. Available: https://www.flightradar24.com/blog/what-are-squawk-codes/

[44] (2015). *Air Traffic Control, Federal Aviation Administration (FAA)*. Accessed: Oct. 24, 2025. [Online]. Available: https://www.faa.gov/documentlibrary/media/order/atc.pdf

[45] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978, doi: 10.1145/359657.359659.

[46] B. Blanchet. (2023). *ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual*. [Online]. Available: https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf

MARCO DE VINCENZI is currently pursuing the Ph.D. degree in computer science with the University of Pisa. He is a Researcher with Italian National Research Center (CNR). He was a Visiting Student with the AUTO-ID Laboratory, Massachusetts Institute of Technology, and the Prof. Suo's Laboratory, Arizona State University. He spent six years in the automotive industry. His research interests include security and privacy in automotive, in particular authentication processes.

CHIARA BODEI received the Ph.D. degree in 2000. She has been an Associate Professor of computer science with the University of Pisa, since 2005. Her research interests include the theory of concurrency and the security of distributed systems, networks, and the Internet of Things. In particular, her work focuses on the application of formal methods to the modeling and analysis of distributed systems, including IoT environments. To this aim, she has worked extensively with process algebras and control flow analysis techniques. More recently, her research has turned to automotive cybersecurity, with particular attention to authentication processes.

GABRIELE COSTA received the master's and Ph.D. degrees in computer science, in 2007 and 2011, respectively. He is an Associate Professor with the SySMA Group, IMT School for Advanced Studies, and a member of the Cybersecurity National Laboratory, CINI. He was part of the Cybersecurity Group, Institute of Informatics and Telematics of CNR; and a Visiting Researcher with ETH Zürich, from 2016 to 2017. He has co-founded the Computer Security Laboratory (CSec), DIBRIS, University of Genova. He is also the Co-Founder of Talos and InLeSi, two startups focusing on cybersecurity.

ILARIA MATTEUCCI received the M.Sc. degree in 2003 and the Ph.D. degree in 2008. She is a Senior Researcher with the Trust, Security and Privacy Group, Institute of Informatics and Telematics of CNR, Italy. Currently, she focuses on automotive cybersecurity, addressing both defensive and offensive aspects. Her work includes analyzing the security properties of the CAN-bus protocol and investigating vulnerabilities in in-vehicle networks and ITS. She actively participates in national and European research projects in the field of information security. Her main research interests include formal methods for secure systems and the analysis of data sharing and privacy policies.