Facing Emerging Challenges in Connected Vehicles: A Formally Proven, Legislation Compliant, and Post-Quantum Ready Security Protocol

Rémi Adelin, Cyrius Nugier, Éric Alata, Vincent Nicomette, Vincent Migliore, Mohamed Kaâniche

Facing Emerging Challenges in Connected Vehicles: A Formally Proven, Legislation Compliant, and Post-Quantum Ready Security Protocol

Rémi Adelin · Cyrius Nugier · Éric Alata · Vincent Nicomette · Vincent Migliore · Mohamed Kaâniche

Abstract Modern vehicles are expected to integrate a variety of connectivity features to enrich safety, entertainment, and driver comfort. This connectivity raises confidentiality and privacy concerns with the risk for the driver to lose control on his data. As vehicles are intended to be used for several years, a major challenge is also to design stable but flexible solutions that can withstand changes in legislation as well as advances in cryptography. Legal frameworks are currently being investigated and implemented to regulate the use of drivers' and vehicles' private information. However, the transcription of these regulations in practice remains an open problem.

In this paper, the first formally proven security protocol for connected vehicles is proposed. It enforces a fined-grained access control policy while providing the flexibility to support recent schemes resistant to a quantum adversary. Its detailed security analysis is assessed using the ProVerif formal verification tool. In addition, a method to generate the access control policy in compliance with the laws is proposed along with an illustrating use case. The method supports both legislation and driver access control to data. Finally, a performance evaluation of the security protocol is provided.

LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France E-mail: {remi.adelin, cyrius.nugier, eric.alata, vincent.nicomette, vincent.migliore}@laas.fr
LAAS-CNRS, CNRS, Toulouse, France E-mail: {mohamed.kaaniche}@laas.fr

This version of the article has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/s11416-022-00426-1. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms

Keywords Vehicle data protection · Formally proven security protocol · Attribute-Based Encryption · ProVerif · Legislation compliant · Post-quantum ready

1 Introduction

In the near future, vehicles are expected to include more and more connectivity features to improve safety and entertainment and to provide additional services to drivers such as real-time navigation and traffic monitoring. In such scenarios, vehicles are expected to communicate their data to various stakeholders and other vehicles through a data storage center, typically a cloud, that acts as an intermediary with stakeholders and can ease data processing and sharing.

Outsourcing this data raises security concerns, as the storage center may have vulnerabilities that may be exploited by attackers to breach the confidentiality, integrity or availability of data. The storage center may also be honest-but-curious and try to read or take advantage of drivers' data. Therefore, it is essential to design security protocols based on suitable cryptographic schemes to protect the exchange and storage of messages in such a connected vehicle environment.

Moreover, the access control to data emitted by the vehicles is, most of the time, managed by the vehicle manufacturers, who can freely decide which stakeholders can or cannot access specific data and under which conditions. However, it is fundamental that this access control management is performed in compliance with laws and regulations that aim to protect users privacy and allow drivers to keep some control over their data (such as the European General Data Protection Regulation[1] and some specific national laws such as the French

[1] https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

Mobility Orientation Law[2] for instance). Technical solutions must take into account the legislation and, as it is constantly changing, anticipate its evolution.

To enforce fine-grained data access control, some cryptographic schemes have been proposed, among them Attribute-Based Encryption (ABE) seems particularly suitable to this context [17]. Relying on a notion of attributes and access trees, ABE schemes have been designed to integrate access control in the encrypted data such that only the authorized stakeholders (i.e., those who are able to decrypt the ciphertext) can access the data. On the other hand, since the lifespan of vehicles is quite long, about ten years or more, it is important to ensure that the cryptographic algorithms embedded in the vehicles remain secure during this period. In particular, they must be chosen so that they can be easily adapted to resist to upcoming quantum attacks.

Although some papers have proposed the use of ABE schemes in the context of connected vehicles to control access to vehicle data, they suffer from the following limitations: 1) to the best of our knowledge, the propositions do not handle the emerging problem of quantum attackers and relies on vulnerable cryptographic schemes; 2) they do not properly handle the legislation in the access control policy; and 3) these papers are, most of the time, dedicated to the description of cryptographic primitives but do not propose a formally proven security protocol, relying on these primitives.

The purpose of this paper is to handle these limitations. The first formally proven, post-quantum, and legislation friendly protocol for connected vehicles is proposed. It relies on the derivation of an access control policy from the law that allows the driver to have increased control on his data.

The main contributions of the paper are summarized below:

- a novel **formally proven** security protocol for connected vehicles:
  - which enforces a **fine-grained access control**;
  - and is generic enough to include post-quantum algorithms.
- **a method to derive an access control policy from the legislation**, that supports drivers service subscriptions and sharing preferences. This method also supports a **transient** data access for sworn stakeholders in specific situations and is compatible with the proposed protocol, illustrated by means of a typical use case;
- **a performance evaluation of the security protocol** in terms of execution time, memory consumption, and messages size.

[2] https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000039666574

The paper is organized as follows. Section 2 specifies the context of this work and the challenges addressed, the entities and data flow, as well as the attacker model, the security properties, and the access control policy requirements that need to be considered. This section concludes with a summary of the proposal that is designed to fulfill these properties and requirements. Section 3 discusses related work addressing the security of connected vehicles communications. Section 4 recalls some important definitions about the cryptographic schemes on which the security protocol is built. In particular, the use of ABE cryptographic schemes is motivated and discussed in a post-quantum context. Section 5 presents in detail the security protocol and describes the enforcement of the security properties, as well as the different scenarios that compose the protocol. Section 6 presents the protocol formal verification using the ProVerif tool and describes the ProVerif syntax required to specify the properties, the simplification considered to model the protocol, and the protocol verification. Section 7 proposes a method to derive the ABE cryptographic components (attributes used to encrypt the messages as well as the access trees used to generate the decryption keys) according to the legislation, the driver's consent and contracts. This section ends with a use case illustrating this method. Section 8 presents the experiments carried out to assess the performances of the security protocol. Section 9 discusses some remaining challenges of the approach and Section 10 draws conclusions and outlines some directions for future work.

2 Context and problem statement

Currently investigated connected vehicle architectures with enhanced on-board capabilities are designed to regularly send data about the state of the vehicle or of its environment. Nevertheless, an efficient data management system is required to store this data. Conventional architectures (such as [3]) rely on a centralized architecture for the storage center. This data is expected to be used by many stakeholders who have to connect to the storage center to retrieve it. Such a centralized architecture is illustrated in Figure 1. Generally, during communication, data is encapsulated in a message before being sent. Then, each legitimate recipient can de-encapsulate the message to retrieve the data. In the remainder of this paper, an encapsulated data is called a message and a de-encapsulated message is called a data. This section presents the entities involved in the architecture, the data flow, the attacker model, the security

[3] https://www.bmwgroup.com/en/innovation/technologies-and-mobility/cardata.html

properties, and the access control policy requirements to be verified. It concludes with a discussion of relevant security issues of current architectures and the proposal to deal with these issues.

2.1 Entities involved

Existing architectures typically involve four entities: the manufacturer, the vehicles, the storage center, and the stakeholders. The **Manufacturer** of the vehicles (bus, cars, trucks, etc.) is most often considered responsible of the storage center deployment and the access control policy. **Vehicles** are operated by drivers and regularly send data to the storage center, by means of messages. The **Storage Center** stores the messages and delivers them through dedicated queries. It is usually deployed in clouds. **Stakeholders** are connected to the storage center to retrieve the relevant messages through dedicated queries. They can include public services (police, courthouse, etc.) or private companies (IT companies, insurance companies, etc.).

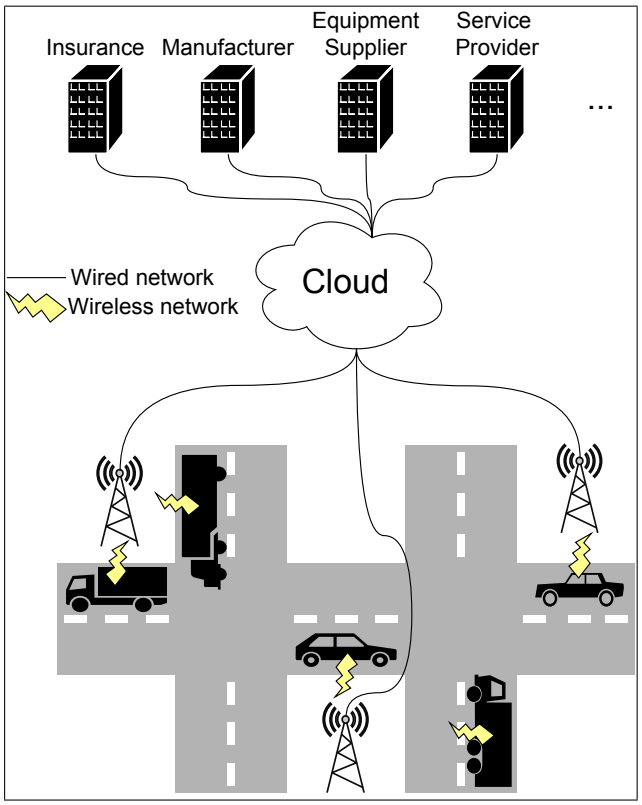

Fig. 1 Conventional architecture

2.2 Data flow

Such architectures must handle three different scenarios. (Scenario **S1**) A vehicle must be able to send messages to the storage center and the storage center has to store these messages. (**S2**) A stakeholder must be able to query the storage center to retrieve the messages it is authorized to read and the storage center has to deliver the corresponding messages. (**S3**) A vehicle must be able to query the storage center to retrieve its own messages and the storage center has to send the corresponding messages back to the vehicle.

2.3 Attacker model

External attackers and honest-but-curious legitimate entities of the system are considered.
An external attacker is an illegitimate participant in the communication. It is assumed that such an attacker has access to all messages exchanged on the network. This pessimistic scenario is purposely considered, though in real situations, an external attacker is unlikely to be able to successfully intercept all messages. The attacker is also supposed to know all the cryptographic schemes used during the communication and all the associated public information (i.e., public keys), but does not have access to secret information (i.e., master and secret keys). The operations that an attacker can perform are specified in the Dolev-Yao model [16]. For instance, he can retrieve a message, modify a message, encrypt a message, inject a message, or perform a man-in-the-middle attack.
An honest-but-curious legitimate participant is supposed to perform its operations correctly, but may try to obtain more information about the messages received legitimately without cooperating with other parties. For instance, a stakeholder or the storage center may try to decrypt a ciphertext it has received even though it is not supposed to be the legitimate recipient of this ciphertext. Such a situation is considered to be similar to an external attacker possessing the secret information of the honest-but-curious participant.
The Dolev-Yao model is limited as it does not consider the leak of secret information. Since a secret information leak cannot be excluded, even if unintentional, the impact of such a leakage must be duly analyzed. The Canetti-Krawczyk model [11] extends the Dolev-Yao model by considering such leaks. In the attacker model of this paper, the Canetti-Krawczyk model is partially taken into account. Actually, the leak of secrets keys enabling to prove the identity of the emitter is not considered, but the leak of the secret keys of either the vehicle, the storage center, or the stakeholder, enabling

the attacker to decrypt the corresponding encrypted messages, is considered and discussed in Section 6.

2.4 Security properties

Security properties are identified in the following, they take into account the attacker model previously described. These are high level properties, which do not consider the implementation details of the architecture. The first properties deal with integrity and authenticity. (**P1**) The legitimate recipient of a message sent by a vehicle must be able to check its integrity and thus detect its potential corruption by an attacker. (**P2**) The storage center must be able to identify whether the messages received come from a legitimate vehicle registered with the manufacturer or not.
The following properties deal with permissions. (**P3**) A vehicle must have the right to access to the data in a message that it has previously sent. Such a property is all the more important as it relates to drivers privacy as stated in the European General Data Protection Regulation. (**P4**) A stakeholder must be able to access to the data in a message for which it has been authorized. The next properties deal with confidentiality. (**P5**) The storage center must not be able to recover the data contained in the messages. (**P6**) A recipient who retrieves a message must not be able to access the included data if it has not been authorized. This property is essential considering that many recent leaks of private information from storage centers have been reported[4].

2.5 Access control policy requirements

As previously stated, an important issue in such security architectures is the derivation of an access control policy complying with the legislation. For that purpose, the following five requirements must be satisfied. (**R1**) The security policy must be compliant with the article of the laws that define a set of permissions and prohibitions regarding some stakeholders. The article 32 of the French Mobility Orientation Law is an example of article that stipulates such rules. Nonetheless, the legislation does not explicitly refer to a member of a company but rather refers to functions or roles of stakeholders. Therefore, from our point of view, the compliance to the legislation seems to be naturally transposed to a role-based approach. (**R2**) Driver's consent must be handled by the access control policy. Each driver can consent to share his data with stakeholders, as long as this choice complies with the legislation. (**R3**) A driver may subscribe to a service by means of a contract that stipulates which type of data must be shared to the stakeholder. Such contracts must be included in the access control policy, as long as they comply with the law. (**R4**) A stakeholder may delegate different permissions to several branch of its internal organization (e.g., departments of an insurance company). This is not explicitly stated in the laws but fundamental from a practical perspective. As such, a delegation mechanism must be included in the access control enforcement. (**R5**) In specific cases, data that were already sent must be accessed by a sworn stakeholder. For instance, in case of an investigation from the police after an accident, the police may require access to the position data of the vehicle before the accident. The access control policy must take into account such sworn stakeholders.

2.6 Discussion and proposal

In traditional architectures (such as [3]) security mechanisms are typically implemented 1) to ensure data confidentiality during communication, against a passive listener on the network; 2) to authenticate a sender (a vehicle) to ensure that a message was sent by a legitimate sender; and 3) to authenticate a reader (a stakeholder) to ensure that it has the rights to access the data. Thus, this architecture covers properties **P1**, **P2**, and **P4**.
However, since data confidentiality is ensured through encrypted communications while the data is stored in plaintext in the storage center, the property **P5** is not satisfied. The storage center is a key component that is responsible for enforcing access control. As the storage center has full control over the plaintext data, a leak or a misbehavior on its part can expose the data to an unintended recipient: the property **P6** is also violated. Finally, conventional architectures do not make the possibility for drivers to be able to access to their own data as a primary objective. Thus, it is necessary to give this possibility to the drivers to ensure property **P3**.
To deal with these issues, this paper proposes a secure protocol aiming to improve conventional architectures and thus ensures the missing properties **P3**, **P5**, and **P6**. This protocol relies on Attribute-Based Encryption and Symmetric Encryption to ensure the confidentiality of data against unintended recipients (including the honest-but-curious storage center) and on Group Signature to authenticate vehicles. The underlying cryptographic schemes are sufficiently generic to adapt to post-quantum algorithms. The security protocol has been formally proven using the ProVerif tool and satisfies the security properties. The paper also proposes a method

[4] https://support.parkmobile.io/hc/en-us/articles/360058639032-Update-Security-Notification-March-2021, https://support.wattpad.com/hc/en-us/articles/360046141392-FAQs-on-the-Recent-Wattpad-Security-Incident

Table 1 Feature comparison of the proposal with other related works, ● represents a satisfied property and ○ represents an unsatisfied property

	Communication	Scheme	Genericity	Protocol	Post-Quantum	Legislation	Automatic protocol verification
[19]	v2v	ABE	Limited genericity	○	○	○	○
[57]	v2v	ABE	Not generic	●	○	○	○
[41]	v2v	ABE	Not generic	●	○	○	○
[25]	v2v	ABE	Not generic	●	○	○	○
[26]	v2v	ABE	Not generic	●	○	○	○
[34]	v2v	ABE	Not generic	●	○	○	○
[24]	v2v	ABE	Not generic	●	○	○	○
[44]	v2v	ABE	Not generic	○	○	○	○
[52]	v2v	IBE	Not generic	○	○	○	○
[23]	v2s	ABE	Not generic	●	○	○	○
[35]	v2s	ABE	Not generic	●	○	○	○
[48]	v2s	IBE	Limited genericity	○	○	○	○
[56]	v2s	IBE	Not generic	○	○	○	○
Our proposal	v2s	ABE	Generic	●	●	●	●

to generate the access control policy in compliance with the law, enabling to satisfy **R1** to **R5** requirements. For that purpose, a trusted authority is included in the architecture, that is in charge of the permissions management for the different stakeholders, in compliance with the law. In traditional architectures, such permissions are generally managed by the manufacturer, who can freely grant access rights to any stakeholders while possibly not respecting the privacy of the driver.

3 Related work

Many papers deal with security and privacy issues due to data outsourcing in Cloud infrastructures and propose suitable cryptographic schemes and protocols. The recent survey from Domingo-Ferrer et al. [17] summarizes these different proposals. In this survey, Identity-Based Encryption (IBE) and Attribute-Based Encryption cryptographic schemes are considered relevant in this context. While IBE represents the authorized recipients using an identity, ABE specifies a set of authorized recipients using attributes. ABE is preferred over IBE in the following as it seems more suitable to enforce a fined-grained access control policy.

This section focuses on papers that specifically deal with data security in connected vehicles communication scenarios. A classification of these papers is presented in Table 1, according to several criteria: 1) **communication**: are the messages emitted from a vehicle to other vehicles (Vehicle-to-Vehicle: v2v) or from a vehicle to stakeholders (Vehicle-to-Stakeholder: v2s)? 2) **scheme**: does the paper rely on ABE or IBE? 3) **genericity**: can the proposed solution be adapted to other cryptographic schemes in the literature? 4) **protocol**: does the paper propose a new cryptographic scheme or a protocol that uses existing schemes? 5) **post-quantum**: does the contribution hold with a quantum adversary? 6) **legislation**: does the paper consider legal constraints from the law? and 7) **automatic protocol verification**: does the paper provide formally verified security guaranties with automatic verification tools? Note that the communication column refers to the message recipient, in particular the communication direction is considered v2v if the destination is a vehicle, even if for some papers the data is temporarily stored on a cloud before reaching its final destination. Similarly, some papers deal with secure communications between a vehicle and a cloud, and only mention the existence of a stakeholder that will process the data to provide a service. These papers are classified in the v2s category.

Most of papers in Table 1 focus on v2v communications security [19,57,41,25,26,34,44,24,52] and mainly deal with the security of messages sent to the vehicles to improve driving, for instance, by anticipating collisions or by warning the driver of traffic jams. Since the legislation establishes rules specifying allowed or prohibited accesses to specific data for specific stakeholders, the approaches focusing on v2v communication security are not appropriate as they do not consider the access control of data sent by the vehicles and consumed by different stakeholders.

Only few papers focus on v2s communications [56,48,23,35] and have some connections with our proposal. The proposals of Zhao et al. [56] and Vaanchig et al. [48] are based on IBE schemes and the protocols proposed by Horng et al. [23] and Luo and Ma [35] rely on ABE.

The two papers by Zhao et al. and Vaanching et al. propose an IBE crytographic scheme to preserve the confidentiality of the data sent by the different vehicles, in which the storage center is not able to access them in plaintext and the different stakeholders can only access the data for which they have been authorized. Nevertheless, these papers mainly propose cryptographic schemes that could be used in a security protocol but do not themselves propose a security protocol.

Horng et al. [23] propose an ABE scheme and a protocol based on this scheme. In addition to addressing data confidentiality, their proposal includes computing nodes located in the Cloud that are used by vehicles to outsource a part of the encryption and decryption process. Their proposal also includes a revocation mechanism and a multi-authority mechanism.

Similarly, Luo and Ma [35] propose an ABE scheme and a protocol with decryption outsourcing and revocation. Additionally, their proposal is multi-authority and the central authority cannot decrypt all ciphertexts without the master secret key of multiple authorities.

Overall, even if these four papers assess the security of their scheme or protocol using manual proofs, the security properties are discussed but not formally proven by means of a formal verification tool. Moreover, these proposals rely on schemes that can be reduced to the discrete logarithm problem and thus are vulnerable to a quantum attacker. Hence, none of the proposals were designed to be post-quantum compliant. In addition, they do not verify whether the emitter of an encrypted data is a legitimate vehicle or not and the authors do not allow a legitimate vehicle to later access some data that it has previously sent to the storage center. Finally, the authors do not explain how the access control management can be enforced, especially to be compliant with the law, which is, from our viewpoint, essential as the communication considers stakeholders supposed to consume the data. The secure protocol and the method to derive the ABE attributes and access trees proposed in this paper aim at tackling these issues.

4 Mathematical background

In this section, the mathematical background needed to introduce the protocol is provided. Three cryptographic primitives are presented: Attribute-Based Encryption, Symmetric Encryption, and Group Signature.

4.1 Attribute-based encryption

4.1.1 A note on post-quantum ABE and implications

Attribute-Based Encryption, introduced in [45], is an asymmetric encryption scheme that can support fine-grained access control policies. Two concepts are used to specify access control: attributes and access trees. Access trees are trees whose internal nodes are AND/OR logical connectors and whose leaves are attributes. Access is granted if the access tree is satisfied by the attributes. Two major constructions of ABE schemes exist: Key-Policy ABE (KP-ABE) [45,22,40] when attributes are set during encryption and Ciphertext-Policy ABE (CP-ABE) [4,55,50,15,53] when the access tree is set during encryption.

ABE is generally constructed from bilinear pairings [10] or lattices [55]. While bilinear pairings-based ABE are subject to strong vulnerabilities against quantum machines, lattice-based ones are considered quantum resistant [20,38,39,42,43]. Historically, ABE is split into two security notions: selective security and adaptive security (also called full security). For a typical adversary/challenger security game, selective security requires access trees (or attributes) to be known before public parameters are generated. This limitation typically restricts the type of access trees achievable by the scheme (or the number of attributes), which limits the expressiveness of the access control policy.

Due to a fairly mature literature on ABE based on bilinear pairings, adaptive security is assumed for all recent schemes with support for the $\mathbf{NC^1}$ class of problems. For lattices, the landscape is more contrasting [9,54]. Achieving adaptive security has been an open problem for many years, and although some recent constructions have achieved the adaptive security, they still have some limitations. In [47], Tsabary et al. proposed the first adaptive post-quantum CP-ABE, but with the restriction that access trees must follow a t-Conjunctive Normal Form (t-CNF), where t is the exact number of literals for the clauses. In [49], Wang et al. proposed an adaptive post-quantum CP-ABE for any polynomial sized circuit, reducing the decryption cost from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$ with public parameters linear with the number of attributes.

4.1.2 ABE algorithms definition

ABE schemes typically consist of four algorithms (identified with the prefix **abe**). In the following, a slight modification of the usual definition of ABE is realized to facilitate the understanding of the protocol. In par-

ticular, KP-ABE is presented as it is the construction used in the protocol:

- **abe_setup(1^λ)**: Given a security parameter λ, generates the master key **MK**. This key will be used to derive both public and secret keys;
- **abe_pkgen(MK)**: Using **MK**, generates the public key **PK**;
- **abe_skgen(T, MK)**: Given an access tree **T**, generates the secret decryption key **SK** associated to **T** using **MK**;
- **abe_enc(μ, X, PK)**: For attributes **X**, encrypts message μ using **PK**, outputs the ciphertext **C**;
- **abe_dec(C, SK)**: For a ciphertext **C**, decrypts **C** using **SK**, outputs μ only and only if the attributes chosen during encryption match the access tree chosen during **SK** generation.

Let us note that an abstract definition of KP-ABE is purposely provided since the protocol is designed to support current and future ABE schemes as much as possible (including post-quantum ones). As long as the ABE scheme securely supports all previously defined algorithms, it can be used in the protocol.

4.2 Symmetric encryption

A Symmetric (S) encryption scheme uses the same symmetric key to encrypt and decrypt messages. A scheme S is composed of a set of three algorithms (identified with the prefix **s**) and the most common standard implementation is AES [14]. An abstraction of this scheme is presented in the following. In a similar way, this abstraction is generic enough to include all S schemes:

- **s_setup(1^λ)**: Given a security parameter λ, generates a secret key **SK**;
- **s_enc(μ, SK)**: Using **SK**, encrypts message μ, outputs ciphertext **C**;
- **s_dec(C, SK)**: Using **SK**, decrypts ciphertext **C**, outputs message μ only and only if the same key **SK** has been used during encryption and decryption.

4.3 Group signature

A Group Signature (GS) scheme [12] allows a member of a group to anonymously generate a signature on behalf of the group. A key issuer is generally defined as the entity responsible for generating and distributing signing keys to legitimate members of the group. A GS is a set of five algorithms (identified with the prefix **gs**). The first post-quantum group signature scheme was proposed in 2010 by Gordon et al. [21]. Most of existing proposals are based on lattices [31,32,33], but some proposals exist for codes [18] and hashes [36]. An abstraction of this scheme is presented in the following and is generic enough to include all GS schemes:

- **gs_setup(1^λ)**: Given a security parameter λ, generates a signature master key **MK**;
- **gs_pkgen(MK)**: Using **MK**, generates the signature public key **PK** used to verify signatures;
- **gs_skgen(ID, MK)**: Using **MK**, generates the signature secret key **SK** associated to identity **ID**;
- **gs_sign(μ, SK)**: Using **SK**, generates the signature **s** for message μ;
- **gs_verif((μ,s), PK)**: Using **PK**, verifies if the signature **s** is a valid signature of message μ. Outputs $\top$ if the signature is valid, $\bot$ otherwise.

5 Secure protocol

This section is dedicated to the detailed presentation of the secure protocol. An overview of the architecture that supports this protocol is first presented, then the security properties enforcement is explained, and finally the protocol scenarios are detailed.

5.1 Architecture overview

The implementation of a secure and trusted control of data from different types of parties (users, legislation, stakeholders, ...) requires a modification of the classical architecture presented in Section 2. In particular, an independent trusted authority is required to define the specific attributes that must be set during data encryption using ABE, with respect to the law, and to manage the different access trees for the stakeholders, also in compliance with the law. Detailed information is provided in Section 7. In this section, the existence of two functions provided by the trusted authority is assumed: the get_attrs function and the get_access_tree function. The **get_attrs** function takes as input either the storage center identity (ID_{sc}) or a vehicle identity (ID_v) and a context (**C**), it provides the attributes used during the ABE encryption. The **get_access_tree** function takes as input an entity identity (either a vehicle, the storage center or a stakeholder), it provides the access tree used during ABE key generation.

5.2 Security properties enforcement

This subsection describes how the security protocol proposed in this paper has been purposely designed so that the **P1** to **P6** security properties are enforced. A signature mechanism is used to enable message integrity

verification (**P1**) and to allow the storage center to verify that a message was issued by an authorized vehicle (**P2**). However, a group signature mechanism is used to avoid overloading the storage center with as many verification keys as legitimate vehicles. To handle this authentication, the trusted authority executes the **gs_setup** and **gs_pkgen** algorithms. Then, it generates the signature secret key noted SIG_SK, using the **gs_skgen** algorithm, for each legitimate sender. A legitimate sender must sign its messages using its secret key. This way, each legitimate receiver can verify the signature using the public key generated by the trusted authority. This secret key provision is assumed to be realized in a secure manner. The signature secret key of a vehicle is embedded in the vehicle, together with associated functions and algorithms, during the manufacturing process. The trusted authority has to provide to each stakeholder the means to allow the access to messages, in accordance with the law (**P4**). The use of KP-ABE is particularly suited as it allows to enforce a fine-grained access control by encrypting the data over a set of attributes that can be only decrypted by the stakeholders possessing the corresponding access trees in their decryption key. Moreover, in this context, KP-ABE schemes are preferable to CP-ABE schemes as they do not require the generation of access trees in the vehicle which would be too costly in such an embedded system with limited resources. This approach also allows the vehicle to have access to its data, as long as it has the appropriate cryptographic materials (**P3**). To generate the corresponding materials, the trusted authority executes the **abe_setup** and **abe_pkgen** algorithms to generate the master key and the public key and the **get_access_tree** function to generate the access tree for an authorized message reader, in compliance with the law. When sending a message, a vehicle must use the **get_attrs** function to get the list of attributes to be set. This function is provided to the vehicle by the trusted authority during the manufacturing process. When no context is given as input, this function simply outputs the identity given as input. The derivation of the access trees according to the law both prevents the storage center (**P5**) and unauthorized recipients (**P6**) from having access to the data in a message.

5.3 Protocol scenarios

5.3.1 Keys generation and distribution

The goal is to generate and to securely deploy the keys for the legitimate entities (vehicles, stakeholders, and storage center). Thus, in a first step, the trusted authority creates the master and public keys for encryption and signature. The public keys are released publicly while the master keys are kept secret. The following notation is used:

Table 2 Protocol symbols

Description	Symbol
Vehicle Nonce	N_v, X_v
Storage center Nonce	N_{sc}, X_{sc}
Stakeholder Nonce	N_{st}, X_{st}
Data	**D**
Context	**C**
Message	M1, M2, M3, M4
Attribute List	L, L1, L2
Query	R
S Ciphertext	C2, C3
S Key	K
ABE Master Key	MK
ABE Public Key	PK
Vehicle ABE SK	SK_v
Storage center ABE SK	SK_{sc}
Stakeholder ABE SK	SK_{st}
ABE Ciphertext	C1, CD
GS Master Key	SIG_MK
GS Public Key	SIG_PK
Vehicle GS SK	SIG_SK_v
Storage center GS SK	SIG_SK_{sc}
Stakeholder GS SK	SIG_SK_{st}
GS Signature	S1, S2, S3, SD

$$\mathrm{MK} = \textbf{abe\_setup}(1^\lambda) \tag{1}$$
$$\mathrm{PK} = \textbf{abe\_pkgen}(\mathrm{MK}) \tag{2}$$
$$\mathrm{SIG\_MK} = \textbf{gs\_setup}(1^\lambda) \tag{3}$$
$$\mathrm{SIG\_PK} = \textbf{gs\_pkgen}(\mathrm{SIG\_MK}) \tag{4}$$

The trusted authority proceeds in the same way for all legitimate entities. For an entity e, it defines its identity ID_e. Then, it generates the access tree A_e using **get_access_tree**, the secret key SK_e, and the signature secret key $\mathrm{SIG\_SK}_e$ for this entity. For this generation, it applies the following operations:

$$\mathrm{A}_e = \textbf{get\_access\_tree}(\mathrm{ID}_e) \tag{5}$$
$$\mathrm{SK}_e = \textbf{abe\_skgen}(\mathrm{A}_e, \mathrm{MK}) \tag{6}$$
$$\mathrm{SIG\_SK}_e = \textbf{gs\_skgen}(\mathrm{ID}_e, \mathrm{SIG\_MK}) \tag{7}$$

These keys are assumed to be sent to the corresponding entity through a secure channel. In particular, for a vehicle, the keys can be deployed in a Hardware Security Module during the manufacturing process. A stakeholder can register to the trusted authority at any time. Obviously, it is assumed that the trusted authority verifies

that this stakeholder has the right to register with this identity. Let us note that the stakeholder has the same cryptographic materials as a vehicle because he also needs to verify if the entity it is communicating with is registered in the system.

*5.3.2 Secure vehicle data send scenario (**S1**)*

This scenario depicts a data sending by a vehicle to the storage center. The exchanges are signed so that the vehicle and the storage center can authenticate each other. A symmetric encryption key is chosen by the vehicle to avoid the systematic use of ABE. The data to be stored is encrypted using the set of attributes L2. Moreover, nonces are also used to prevent replay attacks. The details of this scenario are presented in the following and depicted in Figure 2.

① A vehicle initiates a connection request. This request contains a symmetric key K and a nonce N_v (generated using the **random** function). It is encrypted as C1 using a set of attributes L1 that only allows the storage center to decrypt C1 (i.e., the generation of L1 using **get_attrs** with an empty context). This encrypted request is signed as S1 using the vehicle signature secret key SIG_SK_v. Then, C1 and S1 are sent to the storage center as the message M1.

② The storage center receives message M1. First, it verifies the signature S1 using the signature public key SIG_PK. If the signature is wrong, the storage center resets the communication. Otherwise, after extracting the encrypted request C1 from the message M1 (using the **get_msg** function), it decrypts C1 using its secret key SK_{sc}. It obtains the symmetric key K and the nonce N_v sent by the vehicle. The storage center generates a connection response that contains the nonce N_v and a new nonce N_{sc}. This response is encrypted as C2 using K which is signed as S2 using the signature secret key SIG_SK_{sc}. Then, C2 and S2 are sent back to the vehicle as the message M2.

③ The vehicle receives message M2. First, it verifies the signature using the signature public key SIG_PK. If the signature is wrong, the vehicle resets the communication. Otherwise, it extracts the encrypted response C2 and decrypts it using the symmetric key K. It obtains the two nonces and verifies that the nonce N_v has been returned correctly. Then, it gets the data **D** and the context **C** (using the **retreive_data** function). The context and the identifier of the vehicle are used to derive the set of attributes L2. The data is encrypted as C3 using this set and then encrypted again, together with N_{sc}, using K. This way, the storage center will be able to decrypt and retrieve the encrypted data. The vehicle signs the ciphertext as S3 using its signature secret key. At this point, the vehicle successfully sent the data. Finally, C3 and S3 are sent to the storage center as message M3.

④ The storage center receives message M3. In a similar manner, it verifies the signature. It extracts the message and uses K for decryption. It obtains the encrypted data and a nonce. If the nonce has the same value as the one it has generated during ② then it is able to store the encrypted data in its database (using the **db_store** function).

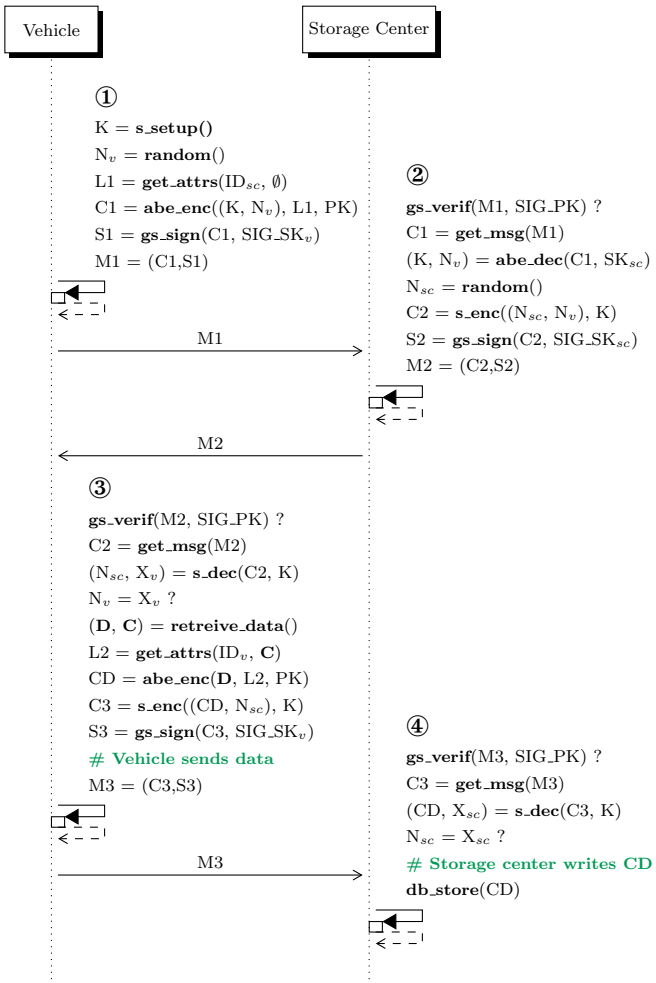

Fig. 2 Secure vehicle data send sequence diagram

*5.3.3 Secure data read scenario (**S2** and **S3**)*

The scenarios **S2** and **S3** differ only regarding the entity that wants to read the data: either a stakeholder or a vehicle. Thus, in the following, only the scenario **S2** is considered, this scenario depicts a data retrieved by a stakeholder from the storage center. Note that the query language used to retrieve data is out of the scope of this paper. The details of the **S2** scenario are presented in the following and depicted in Figure 3. As this scenario uses

the same method as **S1** to prevent replay attacks and to ensure authentication, steps ①, ②, and the beginning of step ③ are similar to the ones in scenario **S1**. In the end of the step ③, the stakeholder generates its query R (using the **gen_query** function), encrypts this query with the key K and sends the encrypted query to the storage center together with its signature. Let us note again that nonces are used to prevent replay attacks.
④ The storage center receives the message M3. It verifies the signature, extracts the message and uses K for decryption. It obtains the query and a nonce. It checks the value of the nonce, which must be identical to the one it has generated during ②. Then it queries its database (using the **db_read** function). The result is sent to the stakeholder together with its signature.
⑤ The stakeholder receives the message M4. It verifies the signature, extracts the message, and uses SK_{st} for decryption. It obtains the result of its query. At this moment, the stakeholder can successfully read the data.

6 Formal verification

Several studies have shown the existence of weaknesses in formally defined protocols (for instance [30] and [13]), thus a formal description of a protocol is a first good step but not sufficient by itself. As connected vehicles have a long lifespan, it is necessary to formally verify the security properties to ensure the viability of the proposed protocol in the long term. This section presents the ProVerif tool used to verify the properties, the simplification realized to model the protocol, and the presentation of the security properties along with their verification.

6.1 ProVerif

Several tools can be used to formally verify the security properties of a protocol (AVISPA [2], ProVerif [8,7], YAPA [3], TAMARIN [37]). ProVerif was chosen as it is stable, mature, still maintained, successful [5,6], and it supports any number of sessions (which is useful to avoid imposing a number of vehicles and stakeholders). In addition, as cryptographic primitives can be represented by an equational theory or rewrite rules, this allows the proof to be established for families of primitives (i.e., as long as the chosen primitive respects the equations).
Considering the attacker model discussed in Section 2.3, ProVerif is well suited as it allows to represent a public channel and it uses the Dolev-Yao attack model [16].
The representation of the protocol in ProVerif is not complex as long as the sequence diagrams are provided (see Figures 2 and 3). This representation is available

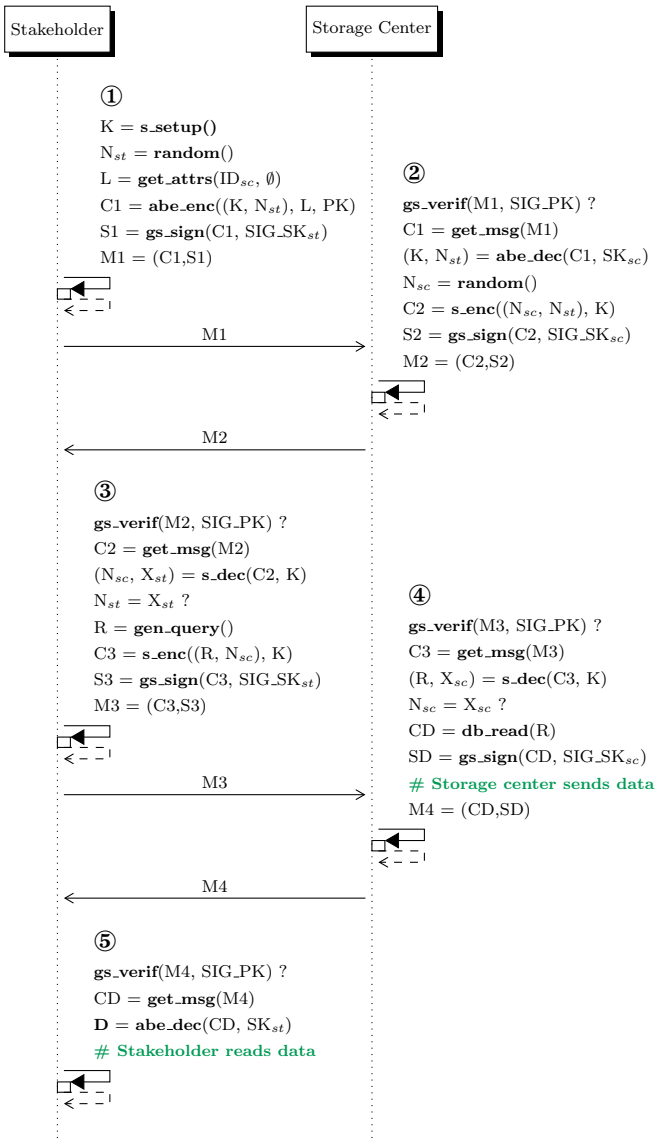

Fig. 3 Secure stakeholder data read sequence diagram

in the appendix, together with the table explaining the syntax of ProVerif. However, it is worth explaining the events and syntax of the queries themselves, which represent the properties to be checked.
The ProVerif code can be enriched with events that can occur several times. In the model, events correspond to a particular step in the protocol. To distinguish them, if necessary, an event can have parameters. For example, an event generated during the execution of the vehicle part of the protocol can have as a parameter the identity of this vehicle. These events correspond to the comments in green in Figures 2 and 3.
During the protocol verification, a key leak is also simulated to analyze the attacker's possibilities and events are also generated when a key leak is forced. The list of considered events is as follows:

- The `vehicle_send` event occurs just before a vehicle sends, on the public channel, a message that contains data (see step ③ in Figure 2);
- The `storage_write` event occurs just before the storage center stores a message in its database (see step ④ in Figure 2);
- The `storage_send` event occurs just before the storage center sends, on the public channel, a message that contains data (see step ④ in Figure 3);
- The `stakeholder_read` event occurs just after a stakeholder retrieves and successfully decrypts a message that contains data (see step ⑤ in Figure 3);
- The `vehicle_read` event occurs just after a vehicle retrieves and successfully decrypts a message that contains data (see step ⑤ in Figure 3);
- The `vehicle_leak` event occurs when the secret key of a vehicle leaks;
- The `storage_leak` event occurs when the secret key of the storage center leaks;
- The `stakeholder_leak` occurs when the secret key of a stakeholder leaks.

The queries used to prove security properties rely on five main constructions:

- `not (event X)`: the result of this query is true if the event `X` is never generated with the protocol. Otherwise, it may be generated and ProVerif provides a trace of its execution. This kind of query is useful to check that all interesting parts of the protocol are reachable;
- `not attacker.pN(V)`: the result of this query is true if the attacker does not have the possibility to know the value `V`. `N` is used to indicate a phase in the protocol. Phases are used only to simplify the representation of the protocol (the first phase corresponds to the keys generation and distribution);
- `secret V1(,Vi)*`: the result of this query is true if all `Vi` values are kept secret. This query is close to the previous one. It is used when the values `Vi` are inner values of a ProVerif process;
- `inj-event(X1) ==> inj-event(X2)`: the result of this query is true if each event `X1` corresponds to a distinct and previous event `X2`;
- `S1 (&& Si)* ==> C1 (|| Ci)*`: the result of this query is true if all the sub-queries `Si` are verified, then one of the conditions `Ci` is true.

6.2 Simplification of the protocol

When integrating the technical details of the protocol into ProVerif, a slight modification of the protocol needs to be realized for the sake of the verification process. First, the implementation of attributes in ProVerif needs to be adapted. The naive method would be to implement unbounded list of attributes directly, but this would lead to an infinite loop during verification. In practice, the number of lists of attributes used for a finite trace is necessarily finite. Thus, the list of attributes that the vehicles can use during the trace is chosen randomly for encryption. This set (of lists of attributes) is selected at the key generation and distribution phase. Then, a distinct lock is associated to each attributes and entities are randomly associated to the locks. In this way, entities that are associated to the same lock can decrypt all messages encrypted with the corresponding list of attribute. Second, some details are omitted in the model: the authentication part during the query of the storage center is not described in ProVerif. In other words, the nonces and the symmetric key are not used during the query scenario. This choice results in a stronger attacker, but it does not affect the security as it will be shown in Section 6.3.

6.3 Security properties and verification

This subsection first presents the considered queries and their link with the properties, followed by the considered leak scenarios and the results of the assessments.
The verification of properties relies on the set of queries from Table 3. The queries **Q1** to **Q5** are used to verify that the protocol is functional by ensuring that all events can occur. Queries **Q6** to **Q10** ensure the secrecy of master keys (for the trusted authority) and secret keys (for vehicles, the storage center, and stakeholders). Query **Q11** ensures that the attacker cannot read the data contained in a message. The queries **Q12** and **Q13** (resp. **Q14** and **Q15**) ensure that, if a vehicle (resp. a stakeholder) is able to read a data from a message, then this message is necessarily stored within the storage center and this message has been sent by a registered vehicle. Query **Q16** ensures that, if a storage center stores a message, this message has been sent by a registered vehicle. Query **Q17** verifies that, if an attacker is able to read a data, this necessarily implies that a secret key has been leaked and that the attacker cannot read more data than this key allows.
These queries together participate in the verification of the properties presented in Section 2.4. The verification of the property **P1** is deduced from queries **Q12** to **Q15**. The verification of the property **P2** is deduced from the query **Q16**. The property **P3** is deduced from queries **Q12** and **Q13**. The vehicle can read messages but only those it has sent. The verification of the property **P4** is deduced from queries **Q14** and **Q15**. The stakeholder can read messages but only those stored within the storage center.

Table 3 Security properties and verification results, ● represents a satisfied property and ○ represents an unsatisfied property

Query id	Security property	Expected result	NO_LEAK	STAKEHOLDER_LEAK	VEHICLE_LEAK	STORAGE_LEAK
Q1	`not event(vehicle_send(va_2,conj_5,ct,msg_3))`	false	●	●	●	●
Q2	`not event(vehicle_read(conj_5,ct,msg_3))`	false	●	●	●	●
Q3	`not event(storage_write(ct))`	false	●	●	●	●
Q4	`not event(storage_send(ct))`	false	●	●	●	●
Q5	`not event(stakeholder_read(sap_2,ct,msg_3))`	false	●	●	●	●
Q6	`not attacker_p4(abe_mk[])`	true	●	●	●	●
Q7	`not attacker_p4(gs_mk[])`	true	●	●	●	●
Q8	`secret storage_abe_sk_1,storage_abe_sk`	true	●	●	●	○
Q9	`secret stakeholder_abe_sk`	true	●	○	●	●
Q10	`secret vehicle_abe_sk_1,vehicle_abe_sk`	true	●	●	○	●
Q11	`secret msg_2,msg_1,msg`	true	●	○	○	●
Q12	`event(vehicle_read(va_2,ct,msg_3))` `==> event(vehicle_send(va_2,conj_5,ct,msg_3))`	true	●	●	●	●
Q13	`event(vehicle_read(va_2,ct,msg_3))` `==> event(storage_write(ct))`	true	●	●	●	●
Q14	`event(stakeholder_read(sap_2,ct,msg_3))` `==> event(vehicle_send(va_2,conj_5,ct,msg_3))`	true	●	●	●	●
Q15	`event(stakeholder_read(sap_2,ct,msg_3))` `==> event(storage_write(ct))`	true	●	●	●	●
Q16	`inj-event(storage_write(ct))` `==> inj-event(vehicle_send(va_2,conj_5,ct,msg_3))`	true	●	●	●	●

Q17	`attacker_p4(msg_3)`
	`&& event(storage_write(abe_enc(msg_3,ext_attrs(a_3,va_2),abe_pkgen(abe_mk[]))))`
	`==> event(vehicle_leak(va_2,abe_skgen(vap_2,abe_mk[])))`
	`|| (event(stakeholder_leak(sap_2,abe_skgen(sap_2,abe_mk[])))`
	`&& event(stakeholder_conj(sap_2,a_3)))`
		true	●	●	●	●

The properties **P5** and **P6** are deduced from the query **Q17**. If the attacker is able to read a message, then the secret key of the corresponding vehicle or the secret key of a stakeholder able to read the message has leaked. Moreover, if the secret key of the storage center has leaked, the attacker has the same privileges as the storage center. Thus, considering the Dolev-Yao model [16], if the attacker is not able to read the data, then the storage center is not able to read the data either and the property **P5** is checked.

Four campaigns have been carried out: one with no leaks and one for each type of leak (a stakeholder secret key, a vehicle secret key, and the storage center secret key). The results are shown in Table 3. The first column indicates the identifier of the query, the second column details the queries and the third column indicates the expected result of the queries evaluation. The last columns correspond to the queries evaluation for the different campaigns: a full circle indicates that the query is valid (corresponds to the expectations) otherwise the symbol is an empty circle.

This table shows that, first of all, without any leak, all queries are validated. Thus, all properties are satisfied and the minimum expected for such an architecture is provided. Moreover, it is formally verified that the storage center is not able to read the data contained in a massage and the attacker cannot either. In case of a leak, five queries are invalidated. In case of a leak of a stakeholder secret key, the first empty circle is obvious as it indicates that the attacker has obtained

this secret key. The second empty circle indicates that the attacker is able to read the messages but the last query guarantees that these messages are those that the stakeholder was already able to read. In case of a vehicle secret key leak, the empty circles have a similar meaning as the previous campaign. In case of a storage center secret key leak, the only information the attacker is able to obtain is its secret key and the attacker is not able to read any messages. Thus, this last campaign gives us the guarantee that the storage center is not able to read data.

7 Legislation-compliant access control

In the previously described protocol, messages sent by vehicles are encrypted over a set of attributes and the stakeholders secret keys used to decrypt these messages are produced using an access tree. These attributes and access trees must be generated with respect to the law, thus this section describes a method for generating them in compliance with the legislation. This method satisfies the **R1** to **R5** requirements presented in Section 2.5 and this section also presents how these requirements are actually satisfied.

7.1 Overview

To cover the different aspects of the legislation, an attribute can be a role or an identity, (so-called role/identity in the rest of the paper), a contextual information or a stakeholder-defined string:

- *a role/identity attribute* identifies stakeholders or vehicles authorized to read the content of a message either by their identity or, for a stakeholder, by their role;
- *a contextual attribute* identifies either the vehicle position, or the data sending date, or the data type (e.g., engine temperature), or additional contextual information for specific situations (e.g., accident). Such attributes are used to authorize stakeholders to access to the data under context related conditions (i.e., at a precise time, in a specific location, or in case of an accident);
- *a stakeholder-defined attribute* is an attribute defined by a stakeholder during delegation.

Access trees are provided by the trusted authority, or can be delegated by a stakeholder to another stakeholder (**R4**). The trusted authority uses the `get_access_tree` function to produce the access tree associated to the storage center, or a stakeholder, or a vehicle. Vehicles access trees are provided into the vehicle during their manufacturing and only include the vehicle identity. Similarly, the storage center access tree includes only its identity. Stakeholders access trees embed *role/identity attributes* and *contextual attributes*, and in case of delegation, they also embed *stakeholder-defined attributes*. The choice of *role/identity attributes* and *contextual attributes* is realized in accordance with the law (**R1**). In specific situation, a stakeholder can be sworn in by a legal organization to access to data previously sent through the use of a transient access tree, as explained in Section 7.3 (**R5**). The remaining of this section focuses on stakeholders access trees.

The algorithm that identifies the attributes for every data transmitted by the vehicle must be designed in compliance with the law and must not be under the control of the vehicle manufacturer (**R1**). As such, the trusted authority derives the so-called *law attributes* and each manufacturer has to embed this set during the implementation of its vehicles. For instance, if the law stipulates in case of an accident, the data recorded in the vehicle at the time of the accident must be available to the police, then the corresponding attributes must be set during the encryption process so that the police can decipher the corresponding messages using the access tree embedded in their secret key, and this decision does not belong to the manufacturer, nor the driver (see Section 7.4.1). Additionally, the driver may also set attributes during the encryption process as long as these attributes (so-called *driver attributes*) comply with the law. The two situations in which the driver may add these attributes are described in details in Section 7.4.2: 1) the driver consents to share its data with a stakeholder (**R2**); 2) the driver has signed a contract with a stakeholder (**R3**). In both situations, the added attributes may target a stakeholder with a delegated key (**R4**). During encryption, the vehicle automatically extracts the *law attributes* and *driver attributes* matching the current context and type of the data that is to be sent and also sets the *contextual attributes*.

Figure 4 and 5 summarize the main steps of the attributes and access trees generation process (the different steps are detailed in the remaining of this section). In these figures, a rectangle corresponds to an entity (either a vehicle, a stakeholder, the storage center, or the trusted authority); an ellipse corresponds to a data; a rounded rectangle corresponds to the execution of a process; an arrow corresponds to the transfer of a data; an hexagon corresponds to the execution of a specific function; a diamond corresponds to an exclusive choice over its input arrow; a fully rounded rectangle corresponds to the internal parameter of a function; and a dash arrow corresponds to a transfer of data used as an internal parameter of a function.

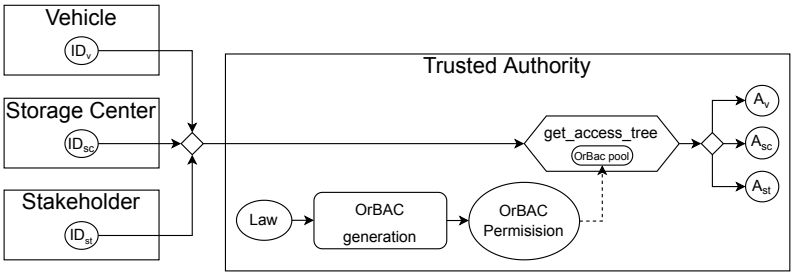

Fig. 4 Access trees generation method

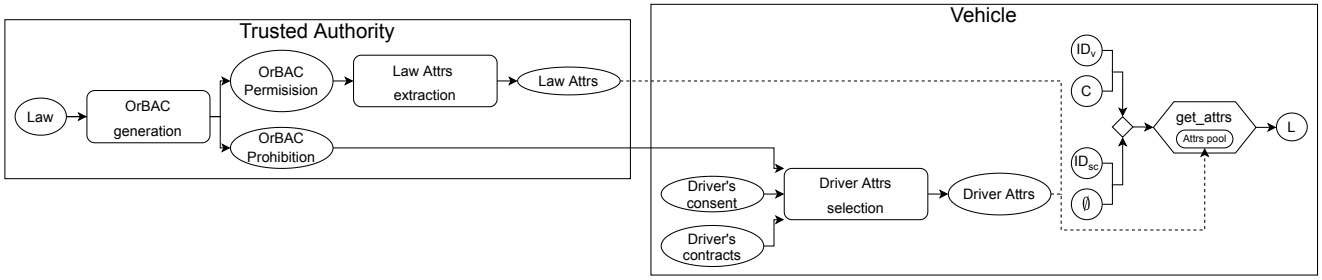

Fig. 5 Attributes generation method

7.2 From articles of the law to OrBAC security rules

The derivation of access trees and attributes from the law is a two step process. The first step consists in generating access control rules in a specific formalism from the different articles of the law that are written in natural language. The formalism proposed in this paper to express these rules is taken from the Organization-Based Access Control (OrBAC) model. This step is quite difficult to automate and it is considered to be done manually only once by a competent person, such as a lawyer. In case the laws exhibit some conflicts, such as two different rules authorizing and preventing a stakeholder to access to a same data, the lawyer is considered able to solve these conflicts during this step. The conflict solving process is out of the scope of this article. The second step consists in obtaining the attributes and access trees from the OrBAC security rules. This step can easily be automated by parsing the OrBAC rules. This two step approach is proposed as generating in one step the ABE access trees and attributes would require specific skills that a lawyer is not supposed to have. The generation of OrBAC rules seems to be an acceptable tradeoff as this formalism is quite simple to understand. This subsection describes the first step of the process which corresponds to the `OrBAC generation` process in Figure 4 and 5.

7.2.1 OrBAC formalism

Role-Based Access Control (RBAC) [46] is a flexible access control model in which roles are assigned to users, permissions are assigned to roles, and users acquire permissions by playing roles. The OrBAC [29] model is an extension of RBAC that details permissions while remaining implementation independent. The main idea is to express the security policy with abstract entities only, and thus to completely separate the representation of the security policy from its implementation. Indeed, OrBAC is based on *roles, views, activities* to structure subjects, objects, and actions. In OrBAC, an *organization* is a structured group of active entities, in which subjects play specific *roles*. An *activity* is a group of one or more actions, a *view* is a group of one or more objects, and a *context* is a specific situation that condition the validity of a rule. Actually, the *role* entity is used to structure the link between subjects and organizations. Similarly, objects that satisfy a common property are specified through *views*, and *activities* are used to abstract actions. OrBAC includes four relationships to express the relations between *organizations*, *roles*, *views*, *activities*, and *contexts*: the `Obligation`, `Permission`, `Prohibition`, and `Recommendation` relationships. Security rules in the OrBAC formalism are expressed as follows:

- `Obligation(O, R, V, A, C)`;
- `Permission(O, R, V, A, C)`;
- `Prohibition(O, R, V, A, C)`;
- `Recommendation(O, R, V, A, C)`.

These expressions mean that, in the context `C`, organization `O` grants role `R` the obligation (or the permission, the prohibition, the recommendation) to perform activity `A` on view `V`.

7.2.2 Extracting meaningful information from the law

The objective is thus to generate such OrBAC security rules from the articles of the law. For this purpose, each article of the law is manually parsed and the relevant information to generate OrBAC security rules is retrieved. The articles of the law stipulate situations (which are used to define the OrBAC context `C`), in which data of a certain type (which is used to define the OrBAC view `V`), *can* or *cannot* (which is used to define the type of OrBAC rules, either `Permission` or `Prohibition`) be accessed by any stakeholder fulfilling a certain role (which is used to define the OrBAC role `R`). When no context is provided, the symbol * is used to represent any context. Each OrBAC rule also defines an organization `O` and an activity `A`. In our case, the `O` corresponds to the unique organization considered in this paper which is the trusted authority (TA), and the activity `A` corresponds to the unique activity considered, which is the *read* activity. Let us note that, as the law identifies possible or prohibited access to data, only the `Permission` and `Prohibition` relationships are considered in the following (the `Obligation` and `Recommendation` relationships are not used in this method).
Let us illustrate this step with the article 32 of the French Mobility Orientation Law. Paragraph 32(I)(2), translated in English, states:

> In the **event of a road accident**, make data from **accident data recording** devices and driving delegation data recorded in the period preceding the accident accessible to **officers and agents of the judicial police** for the purpose of determining liability, as well as to the bodies responsible for the technical and safety investigations provided in Article L. 1621-2 of the Transport Code.

Using this example, the following OrBAC rule is generated:

```
Permission(TA, police_force, accident, read,
           accident)
```

7.3 Access trees generation

7.3.1 Access trees generation from OrBAC security rules

This step corresponds to the execution of the `get_access_tree` function in Figure 4. This function takes an identity as input and behaves as follows. If the identity given as input is a vehicle or the storage center identity, the resulting access tree contains only one node which is the input identity. Otherwise, the identity given as input is a stakeholder identity. In such a case, the OrBAC rules are filtered to retrieve only the `Permission` rules that match the role of the stakeholder (i.e., the rules for which the `R` field matches the role of the stakeholder). For each rule, the trusted authority extracts the data type (`V` field of the OrBAC rule) and the contextual information (`C` field of the OrBAC rule) and generates a so-called *role-string* resulting from the logical AND of this information and the role considered: `C AND V AND R`. The final access tree associated to a stakeholder is the logical OR of his *role-strings* and his identity: `(C1 AND V1 AND R) OR ... OR (Cn AND Vn AND R) OR IDst`.

7.3.2 Transient access trees for sworn stakeholders

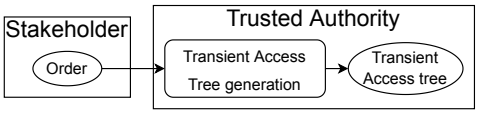

Fig. 6 Transient access tree generation method

Some specific stakeholders can be sworn in by legal organizations. When a stakeholder is sworn in, it receives an order from a legal organization specifying which data it can access. The sworn stakeholder can ask the trusted authority a secret key to access to the information specified in the order. This access tree is a logical AND linking contextual information such as, for instance, a start date, an end date, a data type, and a position. The trusted authority generates the corresponding secret keys and grants it to the sworn stakeholder (as summarized in Figure 6).

7.3.3 Access tree delegation

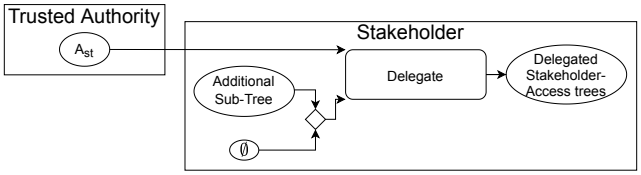

Fig. 7 Delegated access tree generation method

A stakeholder may desire to delegate its access to some specific data to another stakeholder. During delegation,

the resulting secret key is produced with a more restrictive access tree than the original one and may include *stakeholder-defined attributes*. Such attributes are used to characterize the stakeholders who benefit from a delegated key. Stakeholders are assumed to be able to delegate a key to other stakeholders with whom they have a legal relationship such as a subsidiary, a subcontractor, an association, etc. All steps are summarized in Figure 7. Two cases of a more restrictive key delegation are considered:

1. Adding a child to an AND node. In this case, more attributes are required to satisfy the node. The additional child is a sub-tree containing *stakeholder-defined attributes*;
2. Removing a child of an OR node. In this case, less combination of attributes enable to satisfy the node.

7.4 Attributes generation

The attributes used to encrypt the messages to be sent are selected using the `get_attrs` function (represented in Figure 5). This function takes as input an identity and the context of the data. In case of a vehicle identity, the context is used to 1) select attributes from two attribute sets (*law attributes* and *driver attributes*) and 2) to generate the *contextual attributes*. An attribute corresponding to the vehicle identity is also generated to enable the sending vehicle to access to the data it has sent. In case of the storage center identity, an empty context is given as input, and only one attribute corresponding to the storage center identity is forwarded as output. The following subsections describe in details the processes aiming at generating the attributes.

7.4.1 From OrBAC Permission rules to law attributes

This step corresponds to the `Law Attrs extraction` process in Figure 5. The trusted authority relies on the `Permission` rules to generate the *law attributes*. Generating this set is quite easy once the OrBAC security rules are established. The extraction of these attributes can be informally described as follows. For each rule, the data type (`V`), the context (`C`), and the role (`R`) are extracted and a couple `({V,C},R)` is generated. The set of all couples is the law attribute set. This set is included in the vehicle during its manufacturing and must be updated when the laws are themselves modified. During a data encryption, the role is extracted from each *law attribute* that matches the type and the context of the data currently sent. The set of attributes used for encryption is simply the set of extracted roles from the *law attributes*.

7.4.2 Driver control on the attributes

At any time, the driver may add attributes for the ABE encryption of data, as long as they do not violate an OrBAC `Prohibition` rule. This may happen in two situations. First, the driver may consent to open his data to additional stakeholders and may decide to add *roles/identities attributes* corresponding to stakeholders of his choice. He may also add *stakeholder-defined attributes* to share his data with a stakeholder having a delegated key. Second, when a driver signed a contract with a particular stakeholder, the attributes corresponding to this contract must be added at encryption time. Such attributes may simply be the identity of the corresponding stakeholder or some *stakeholder-defined attributes* that are stipulated in the contract. In both cases, couples are formed specifying a context and data type that must be satisfied to generate the corresponding attributes. In the first case, the context and data type are specified by the driver and in the second case they are specified in the contract. During a data encryption, either the *role/identity* or the *stakeholder-defined attribute* is extracted from each couple that matches the type and the context of the data currently sent. The set of attributes used for encryption is simply the set of extracted *roles/identities* or *stakeholder-defined attributes* from the couples. This process corresponds to the `Driver Attrs selection` process in Figure 5.

Role/identity attributes When a driver wants to add *role/identity attributes* or when a contract specifies an *identity attribute*, this must be done in compliance with the law. If the considered data are not covered by any article of the law and thus by any OrBAC rule, then the driver may use any *role/identity attribute*. Otherwise, the considered data are covered by a matching OrBAC `Prohibition` rule (i.e., matching the role, the data type, and context), the driver/contract may only add attributes that do not conflict with such rules. Two situations can be considered:

1. The additional attribute is a stakeholder role, this role must not appear in any OrBAC matching `Prohibition` rules;
2. The additional attribute is a stakeholder identity, the role associated to the stakeholder must not be included in any OrBAC matching `Prohibition` rules. For that purpose, a function that is able, from a stakeholder identity, to provide its role, is also embedded in the vehicle and regularly updated.

Stakeholder-defined attributes They may be freely added by a driver or through a contract. These attributes only

concern delegated stakeholders and the drivers are assumed to know the attributes required to grant access to a delegated stakeholder.

7.5 Requirements satisfaction

This subsection summarizes how the method described in this section satisfies the **R1** to **R5** requirements. The **R1** requirement is satisfied through the generation of access trees and attributes using the OrBAC `Permission` rules extracted from the legislation. The **R2** requirement is satisfied through the use of driver's consent attributes during the `Driver Attrs selection` process and through the respect of OrBAC `Prohibition` rules while providing these attributes. The **R3** requirement is satisfied by taking into account driver's contracts in the `Driver Attrs selection` process and OrBAC `Prohibition` rules while providing these attributes. The **R4** requirement is satisfied through the `Delegate` process generating delegated access trees and the `Driver Attrs selection` process taking into account driver's consent attributes and driver's contracts. Finally, the **R5** requirement is satisfied through the `Transient Access Tree generation` process generating transient access tree and the generation of contextual attributes in the vehicle.

7.6 Use case

This subsection presents a simple use case illustrating the previously described method (i.e., access trees and attributes generation and access control enforcement).

7.6.1 Context

The stakeholders considered are: a meteorological service (`meteo`), two police forces (`policeA` and `policeB`), a road infrastructure manager (`infra`) with three agencies, two in regionA (`infraA1` and `infraA2`), and one in regionB (`infraB1`), an insurance (`insur`) employing a subcontractor (`sc1`) processing speed data only and a subcontractor (`sc2`) processing position data only. The roles of the different stakeholders are as follows:

```
id: meteo, role: meteo_service
id: policeA, role: police_force
id: policeB, role: police_force
id: infra, role: road_infra
id: insur, role: insurance
id: sc1, role: subcontractor
id: sc2, role: subcontractor
```

Each stakeholder is considered as a legal entity. If a stakeholder has multiple agencies, each agency legally depends on the parent company. This applies to the three agencies (i.e. **infraA1**, **infraA2**, and **infraB1**) which depend on **infra**.
One vehicle called `veh` is considered. It is equipped with sensors generating data which types are: speed, pollution, position, temperature, road damage, and accident. To simplify the position representation, the world map is considered to be divided into tiles and a position corresponds to a specific tile, e.g., `tile5`. An accident data is emitted when the vehicle itself has an accident and is able to detect it. Six messages are considered which are sent by `veh` on 07-22-2021, they are presented in Table 4.

Table 4 Messages on 07-22-2021

ID	Time	Tile	Data type
M1	09:55:20	tile5	pollution
M2	09:55:30	tile5	position
M3	09:55:40	tile6	temperature
M4	09:58:05	tile6	road damage
M5	09:59:00	tile6	speed
M6	10:00:00	tile7	accident

7.6.2 Access control specification

As the articles of the law are still in draft form, the proposed articles are inspired from the article 32 of the French Mobility Orientation Law:

L1 Road infrastructure managers can access to the road damage data;
L2 Police forces cannot access to the speed data;
L3 In case of an accident, police forces can access to the accident data;
L4 In case of an accident, a police force can be sworn in and then can access to the position data before the accident.

It is considered that:

- using L4, `policeA` obtains an order to be sworn in to access to the position data before the accident in M6 (i.e., the position data in M2);
- the driver has signed a pay-as-you-drive contract with `insur` which stipulates that the speed data must be accessible by `insur` and `sc1`;
- the driver consents to share the position data with the `road_infra` role and its agencies in regionA, the temperature data with `meteo`, the speed data with the `police_force` role and with `policeB`.

To summarize in term of access control objectives:

Table 5 Access rights matrix representing the access control objectives, ● represents an authorized access and ○ represents a forbidden access

	M1	M2	M3	M4	M5	M6
meteo	○	○	●	○	○	○
policeA	○	●	○	○	○	●
policeB	○	○	○	○	○	●
infra	○	●	○	●	○	○
infraA1	○	●	○	●	○	○
infraA2	○	●	○	●	○	○
infraB1	○	○	○	●	○	○
insur	○	○	○	○	●	○
sc1	○	○	○	○	●	○
sc2	○	○	○	○	○	○
veh	●	●	●	●	●	●

1. the pollution data from M1 must be accessible by nobody, excepts `veh`, as no access control specification identifies the stakeholder that can access a pollution data;
2. the position data from M2 must be accessible by `policeA` since it has been sworn in, and by `infra`, `infraA1`, and `infraA2` as the driver consents to share its position data with `infra` and its agencies in regionA;
3. the temperature data from M3 must be accessible only by `meteo` as the driver consents to share it with `meteo` only;
4. the road damage data from M4 must be accessible by `infra`, `infraA1`, `infraA2`, and `infraB1` as L1 specifies that each road infrastructure manager can access to the road damage data;
5. the speed data from M5 must be accessible by `insur` and `sc1` as it is specified in the pay-as-you-drive contract;
6. the accident data from M6 must be accessible by `policeA` and `policeB` as L3 specifies that the accident data is available to the police forces;
7. all data must be accessible by `veh`.

Let us note that, even if the driver consents to share its speed data with `policeA` and `policeB`, as L2 prevents him to share this data, they must not have the right to access to it. The corresponding access control matrix is represented in Table 5.

7.6.3 Access control enforcement

Attributes list

Each attribute is defined with a prefix and a value.

Stakeholder roles/identities attributes:

- stakeholder roles prefixed with `st_role:`, possible values are `meteo_service`, `police_force`, `road_infra`, `insurance`, and `subcontractor`;
- stakeholder identities prefixed with `st_id:`, the possible values are `meteo`, `policeA`, `policeB`, `infra`, `insur`, `sc1`, and `sc2`.

Contextual attributes:

- the date prefixed with `date:` (e.g., `date:07-22-2021`);
- the hour prefixed with `hour:` (e.g., `hour:08-31`);
- the position prefixed with `position:` (e.g., `position:tile10`);
- the data type prefixed with `type:`, the possible values are `pollution`, `position`, `temperature`, `road_damage`, `speed`, and `accident`;
- a contextual label prefixed with `label:`, the value accident is the only value considered, i.e., `label:accident`.

Stakeholder-defined attributes are prefixed with `st_attr:`, the values considered in this use case are `regionA`, `regionB`, `speed`, and `position`.
Finally, the **vehicle identity** is also an attribute that is prefixed with `v_id:` (i.e., `v_id:veh`).

Vehicle Access tree

The trusted authority generates the vehicle access tree which solely contains the vehicle identity:

```
veh:
    v_id:veh
```

Access trees from the articles of the law

Three OrBAC rules are generated from the articles L1, L2, and L3:

```
Permission(TA, road_infra, road_damage,
           read, *)
Prohibition(TA, police_force, speed,
            read, *)
Permission(TA, police_force, accident,
           read, accident)
```

Let us note that the L4 rule cannot be used to generate OrBAC rules as this rule is dedicated to the sworn in process and is used by the trusted authority when generating transient access trees.
From the OrBAC rules, the following access trees are generated by the trusted authority:

```
policeA:
        (st_role:police_force
        AND type:accident
        AND label:accident)
    OR  st_id:policeA
policeB:
        (st_role:police_force
        AND type:accident
        AND label:accident)
    OR  st_id:policeB
infra:
```

```
        (st_role:road_infra
        AND type:road_damage)
    OR  st_id:infra
meteo:
    st_role:meteo_service
    OR st_id:meteo
insur:
    st_role:insurance
    OR st_id:insur
sc1:
    st_role:subcontractor
    OR st_id:sc1
sc2:
    st_role:subcontractor
    OR st_id:sc2
```

Let us note that, the access tree for stakeholders with no matching OrBAC rules is only composed or their role and identity.

Access trees from delegation

Two cases of delegation are considered: `infra` delegates its secret key to its agencies by adding an AND with the region of its agency and `insur` delegates its secret key to its subcontractors with an AND for each type of data:

```
infraA1/infraA2(delegated):
        (st_role:road_infra
        AND type:road_damage)
    OR  (st_id:infra
        AND st_attr:regionA)
infraB1(delegated):
        (st_role:road_infra
        AND type:road_damage)
    OR  (st_id:infra
        AND st_attr:regionB)
sc1(delegated):
        st_role:insurance
    OR  (st_id:insur
        AND st_attr:speed)
sc2(delegated):
        st_role:insurance
    OR  (st_id:insur
        AND st_attr:position)
```

Transient access tree

As `policeA` is sworn (L4 rule), it obtains a transient key from the trusted authority, due to the occurrence of an accident, containing the following access tree:

```
policeA(transient):
    date:07-22-2021
    AND position:tile5
    AND type:position
    AND hour:09-55
```

Attributes from the articles of the law

In the vehicle, using the OrBAC rules, the following law attributes are generated:

```
((type:accident, label:accident),
  st_role:police_force)
((type:road_damage), st_role:road_infra)
```

Attributes from driver's contract

From the pay-as-you-drive contract, two attributes are added when a speed data is emitted, one specifying the identity `insur` and one specifying the stakeholder-defined attribute `speed`:

```
((type:speed), st_id:insur)
((type:speed), st_attr:speed)
```

Attributes from driver's consent

From driver's consent, `meteo` can access the `temperature` data using its identity and the infrastructure managers in regionA can access the position data using the `infra` identity and the `regionA` stakeholder-defined attribute:

```
((type:temperature), st_id:meteo)
((type:position), st_id:infra)
((type:position), st_attr:regionA)
```

Let us note that no attributes are selected from driver's consent for the access of the `police_force` role and the `policeB` identity to the speed data as the vehicle filters the role and identity using the OrBAC Prohibition rule.

Attributes generation for the data sending events

For the six data sending events, from M1 to M6, the following attributes are selected:

```
M1: (date:07-22-2021, hour:09-55,
    position:tile5, type:pollution,
    v_id:veh)
M2: (st_id:infra, st_attr:regionA,
    date:07-22-2021, hour:09-55,
    position:tile5, type:position,
    v_id:veh)
M3: (st_id:meteo, date:07-22-2021,
    hour:09-55, position:tile6,
    type:temperature, v_id:veh)
M4: (st_role:road_infra, date:07-22-2021,
    hour:09-58, position:tile6,
    type:road_damage, v_id:veh)
M5: (st_id:insur, st_attr:speed,
    date:07-22-2021, hour:09-59,
    position:tile6, type:speed,
    v_id:veh)
M6: (st_role:police_force, date:07-22-2021,
    hour:10-00, position:tile7,
    type:accident, v_id:veh, label:accident)
```

7.6.4 Access control evaluation

Taken into account the attributes associated to the six events and the access trees of the different stakeholders described above, the access control objectives are verified:

1. the pollution data emitted at M1 cannot be accessed by any stakeholder;
2. the position data from M2 is accessible by `policeA` through its transient key, by `infra` through the attribute `st_role:road_infra`, and by `infraA1` and `infraA2` through the attributes `st_role:road_infra` and `st_attr:regionA`;

3. the temperature data from M3 is only accessible by `meteo` through the attribute `st_id:meteo`;
4. the road damage data from M4 is accessible by `infra`, `infraA1`, `infraA2`, and `infraB1` through the attribute `st_role:road_infra`;
5. the speed data from M5 is accessible by `insur` through the attribute `st_id:insur` and by `sc1` through the attributes `st_id:insur` and `st_attr:speed`;
6. the accident data from M6 is accessible by `policeA` and `policeB` through the attributes `st_role:police_force`, `type:accident`, and `label:accident`;
7. all data are accessible by `veh` through the attribute `v_id:veh`.

These results are consistent with the access matrix in Table 5 and show that the attribute and access tree generation method is relevant to enforce the access rights described in the matrix.

8 Performance evaluation

This section is dedicated to the evaluation of the protocol performances with a presentation of the prototype developed to conduct these experiments, the experimental protocol, and the experimentation results regarding the execution time, the memory consumption, and the message size.

8.1 Implementation details

The security protocol is implemented in C language[5], using the KP-ABE algorithms provided by the OpenABE library [51]. This library, developed by Zeutro in C++, relies on the RELIC library [1] to perform the low level cryptographic operations and provides an adapted implementation of KP-ABE large universe construction of [22], which is a pairing based implementation. The symmetric encryption algorithms are implemented using AES in CTR mode from OpenSSL libcrypto library. This mode was chosen as a stream encryption was needed to not expand the size of the ciphertext, others AES modes offering various properties can be used instead of the CTR mode. As a suitable group signature implementation was not available, the signature implementation relies on a classic asymmetric signature from OpenSSL libcrypto library. The signature performs a digest using SHA256 and outputs the signature using ECDSA on the brainpoolP512r1 curve.

[5] Code available at: https://gitlab.laas.fr/jicv_2022_security_protocol/security_protocol

All steps of the scenarios (*Keys generation and distribution*, *Secure vehicle data send* scenario, and *Secure data read* from Subsection 5.3) are implemented as a C function. The message exchanges on the network are not strictly implemented but the impact on the network metrics (network throughput and bandwidth) is evaluated. The get_access_tree function is not implemented as it is executed by the Trusted Authority offline. The get_attrs function is not implemented either as it only performs a search in an array, which has a very limited cost in term of computation time.

The remaining of this section is focused on the *Secure vehicle data send scenario* (Subsection 5.3.2) and more specifically on the steps 1 and 3 which are the most critical steps in term of performances as they are executed on the vehicle, which has limited resources in term of computation power, memory capacity, and bandwidth capacity.

8.2 Experimental protocol

The benchmarks were performed on a Raspberry Pi 3 B+ chosen for its similarity with the resources of a vehicle ECU. It has a 1.4GHz 64-bit quad-core processor (ARMv8) and a 1GB RAM. Three benchmarks were performed: a time benchmark, a size benchmark, and a memory benchmark.

Time benchmark The execution time in millisecond of the steps 1 and 3 is evaluated. The measurements were performed using the processor performance counter and then converted to milliseconds. To reduce the counter variation, the processor was set to its maximum frequency and the benchmark process was isolated on a single core using *cset shield*. This benchmark was performed with a number of attributes varying from 1 to 31 per step of 2. For each number of attributes, the size of the data to be sent in step 3 varies from 1 byte to 4096 bytes per power of 2. Finally, for each combination of number of attributes and message size, 1024 iterations were performed.

Size benchmark The size in bytes of the message produced after the execution of the step 3 is evaluated. It was performed with a number of attributes varying from 1 to 31 per step of 2. For each number of attributes, the size of the data to be sent in step 3 varies from 1 byte to 4096 bytes per power of 2. Finally, for each combination of number of attributes and message size, 16 iterations were performed.

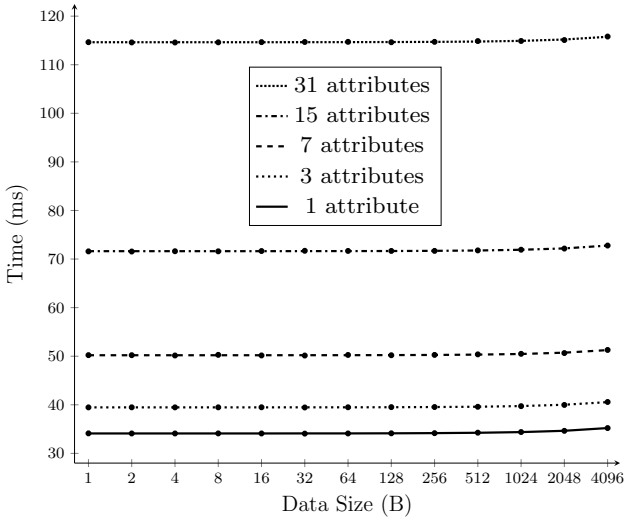

Fig. 8 Step 3 average execution time w.r.t the size of the data for 1, 3, 7, 15, and 31 attributes

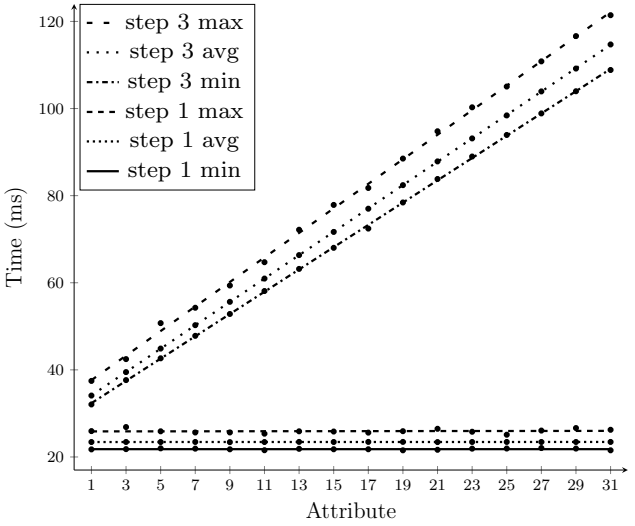

Fig. 9 Execution time w.r.t the number of attribute for a 64 bytes data

Memory benchmark The maximum size in kilobytes of the heap and stack memory regions of step 1 and step 3 is evaluated. The measurements were performed using the Massif tool of Valgrind which profiles the heap and stack size. It was performed with a number of attributes varying from 1 to 31 per step of 2. For each number of attributes, the size of the data to be sent in step 3 varies from 1 byte to 4096 bytes per power of 2. Finally, for each combination of number of attributes and message size, 16 iterations were performed.

8.3 Experimentation results

In the following graphs, the dots represent the data and the curves represent the corresponding linear regression.

8.3.1 Time benchmark

The execution time of step 3 is first evaluated as step 3 regularly performs ABE encryptions and is definitely the most time consuming primitive. Figure 8 represents the average execution time of step 3 for different sizes of data in the case of 1, 3, 7, 15, and 31 attributes. As illustrated in this figure, the execution time is a nearly constant function of the size of the data to be sent. This demonstrates that the size of the data does not have any significant impact on the execution time of step 3. In fact, the OpenABE library implements the ABE encryption algorithm as a Key Encapsulation Mechanism. This algorithm first generates a key that is encrypted with ABE, then the key is used to encrypt the data using AES. As the AES encryption time is low in comparison with the remaining of the operation performed in step 3, the increase in the size of the data has not a significant impact on the total execution time of step 3. In the following, we chose to focus on a 64 bytes data, which is realistic in the context of a data that may be regularly sent by connected vehicles, and on a 4096 bytes data, which represents a worst case scenario.

The impact of the number of ABE attributes on the execution time of step 1 and 3 is evaluated. Table 6 and 7 represent the execution time of step 1 and step 3, respectively in case of a 64 bytes and 4096 bytes data for different number of attributes, and Figure 9 represents the execution time of step 1 and step 3 in case of a 64 bytes data for different number of attributes. These tables and figure show that 1) the execution time of step 1 is nearly constant, which is not surprising as this step is a connection step with only one ABE attribute; 2) the execution time of step 3 linearly increases w.r.t to the number of attributes; and 3) the absolute execution time of step 1 (21.45-27.36 ms) and step 3 (32.06-122.44 ms) remains reasonable in a connected vehicle scenario. Even with 31 attributes, the execution time of step 3 is in average 114.71 ms for a 64 bytes data, which enables approximately 8 step 3 per second, this is perfectly realistic in a real context. Extrapolating on these data, to reach an execution time of one second, the number of attributes should be set to 361. This number of attributes is large enough to include complex scenarios of a real case implementation of the legislation. Furthermore, let us note that the experiments have been carried out on a Raspberry Pi and that the implementation as well as the OpenABE library were not optimized. The execution time would reduce if an optimized implementation and optimized library were used. Overall, these experi-

Table 6 Execution time w.r.t the number of attribute for a 64 bytes data

attribute	step 1 (ms)				step 3 (ms)				step 3 (Hz)
	min	max	avg	med	min	max	avg	med	
1	21.74	25.94	23.43	23.4	32.06	37.46	34.1	34.08	29
3	21.8	26.87	23.44	23.42	37.65	42.47	39.49	39.45	25
5	21.98	25.89	23.43	23.41	42.65	50.73	44.89	44.81	22
7	21.91	25.62	23.45	23.41	47.81	54.26	50.26	50.19	19
9	21.76	25.63	23.41	23.38	52.83	59.37	55.62	55.5	17
11	21.55	25.35	23.43	23.41	58.1	64.72	60.96	60.89	16
13	21.87	25.89	23.46	23.43	63.18	72.17	66.35	66.26	15
15	21.77	25.84	23.44	23.4	68.02	77.87	71.67	71.55	13
17	21.77	25.59	23.46	23.44	72.44	81.76	77.0	76.88	12
19	21.53	25.92	23.46	23.44	78.43	88.51	82.41	82.28	12
21	21.65	26.47	23.42	23.4	83.82	94.78	87.86	87.83	11
23	21.91	25.79	23.44	23.4	88.98	100.28	93.14	93.03	10
25	21.94	25.11	23.45	23.43	93.93	105.05	98.42	98.32	10
27	22.05	26.06	23.44	23.4	98.88	110.83	103.93	103.87	9
29	21.93	26.64	23.44	23.41	103.97	116.63	109.18	109.06	9
31	21.53	26.24	23.45	23.43	108.85	121.43	114.71	114.59	8

Table 7 Execution time w.r.t the number of attribute for a 4096 bytes data

attribute	step 1 (ms)				step 3 (ms)				step 3 (Hz)
	min	max	avg	med	min	max	avg	med	
1	21.67	26.49	23.47	23.41	33.61	38.37	35.21	35.18	28
3	21.79	25.49	23.45	23.45	38.9	43.38	40.58	40.52	24
5	21.97	27.36	23.41	23.38	43.68	50.16	45.89	45.8	21
7	22.06	25.33	23.45	23.4	48.76	56.74	51.29	51.26	19
9	21.62	25.32	23.44	23.42	54.07	60.93	56.67	56.59	17
11	21.93	26.65	23.46	23.43	58.81	66.43	62.02	61.94	16
13	21.88	25.59	23.46	23.42	64.15	72.35	67.37	67.29	14
15	21.85	25.83	23.48	23.45	69.55	79.16	72.78	72.73	13
17	21.54	25.64	23.42	23.4	74.63	83.01	78.14	78.05	12
19	21.45	25.93	23.43	23.4	79.26	88.62	83.49	83.31	11
21	21.99	25.57	23.45	23.43	84.67	94.17	88.88	88.77	11
23	21.97	26.52	23.47	23.45	89.07	102.84	94.26	94.2	10
25	21.72	26.13	23.44	23.42	95.15	106.38	99.64	99.57	10
27	22.06	25.87	23.45	23.43	100.72	111.85	105.11	105	9
29	21.9	27.18	23.46	23.44	105.11	118.26	110.39	110.36	9
31	21.86	25.34	23.45	23.41	109.34	122.44	115.8	115.73	8

ments show that the execution time of the encryption process is acceptable and may not significantly impede the performances of the ECU embedded in the vehicle.

8.3.2 Size benchmark

This benchmark was dedicated to the evaluation of the size of the messages produced by step 3 (M3 in Figure 2), and, as a direct consequence, the network bandwidth required to send the messages. Table 8 and Figure 10 represent the size of the message produced by step 3 for different number of attributes, in case of a 64 bytes and 4096 bytes data, and Figure 11 represents the size of the message produced by step 3 for different data sizes, in case of 1, 3, 7, 15, and 31 attributes. These table and figures show that 1) the message size linearly increases w.r.t to the number of attributes; 2) the message size linearly increases w.r.t to the data size; and 3) for a 64

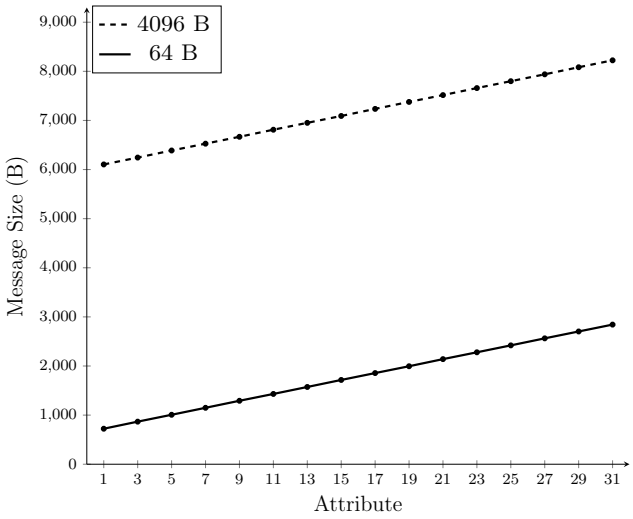

Fig. 10 M3 size w.r.t the number of attribute for a 64 bytes and 4096 bytes data

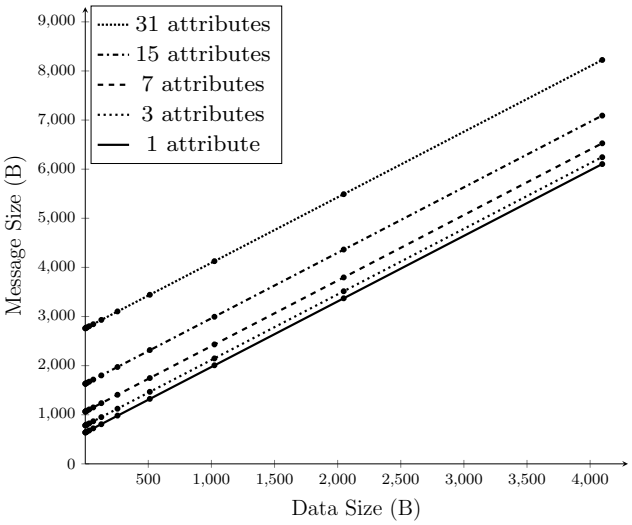

Fig. 11 M3 size w.r.t the data size for 1, 3, 7, 15, and 31 attributes

Table 8 M3 size w.r.t the number of attribute for a 64 bytes and 4096 bytes data

attribute	64 (B)	4096 (B)
1	723	6104
3	867	6244
5	1007	6388
7	1149	6527
9	1292	6667
11	1431	6811
13	1573	6951
15	1716	7091
17	1856	7235
19	1995	7377
21	2140	7516
23	2279	7659
25	2420	7799
27	2563	7939
29	2703	8083
31	2843	8224

bytes data, the cipher expansion is 11.3 for 1 attribute, and 44.4 for 31 attributes. Considering a 64 bytes data with 31 attributes, and, according to the previous experiments, 8 executions of the step 3 per second, the sending of the messages requires approximately 181.95 kbps network bandwidth. The website of the International Telecommunication Union [28] specifies that the 3G provides a minimum speed of 348 kbps for a moving vehicle, which means that the required bandwidth for the protocol can be supported by 3G/4G/5G vehicle connections.

However, with a 4096 bytes data with 31 attributes, the same sending of messages requires approximately 526.34 kbps network bandwidth, which should not be supported by a 3G connection. This estimation was realized by considering that the data are sent at the maximum rate, which may not probably be the case for data of 4096 bytes. Such messages may rather correspond to batch data that are sent less frequently. If a lower sending rate is considered, for example executing step 3 each second with a 4096 bytes data and 31 attributes, the required bandwidth becomes 65.79 kpbs which is compliant with any 3G or 4G or 5G vehicle connection.

To finalize the evaluation of the message size, a worst case is considered in which the vehicle outputs the whole messages that are exchanged on the CAN Bus. The bandwith of the CAN Bus is 1 Mbps [27], so it is considered that a message of 1 Mb with 31 ABE attributes must be sent each second. The required bandwidth for the network connection is then 1.4 Mbps which should not be supported by a 3G connection, but can be supported by 4G or 5G connections.

8.3.3 Memory benchmark

Finally, the maximum heap and stack size required for the execution of step 1 and step 3 is evaluated. Figure 12 represents the average maximum size of the heap and stack of step 1 and step 3, for different number of attributes, in the case of a 64 bytes data. This figure shows that 1) for step 1 and step 3, the maximum stack size is nearly constant with less than 20 kB; 2) for step 1, the maximum heap size is nearly constant and does not exceed 220 kB; and 3) for step 3, the maximum heap size linearly increases w.r.t to the number of attributes and does not exceed 240 kB for attributes between 1 and 31. This size represents less than 0.1% of the whole

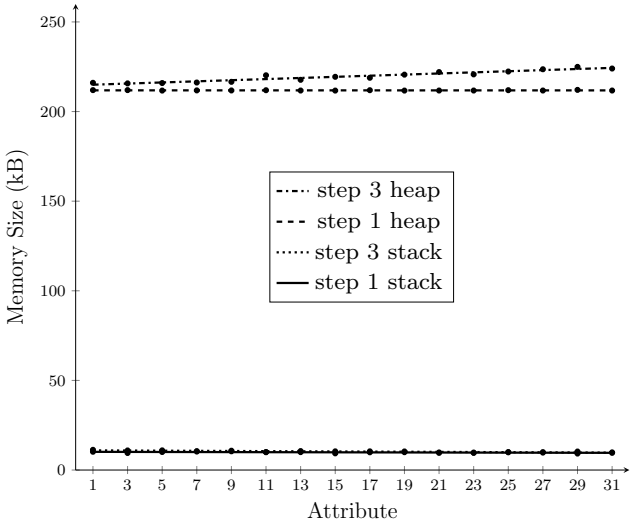

Fig. 12 Step 3 average maximum heap and stack size w.r.t the number of attribute for a 64 bytes data

memory available on the Raspberry Pi, and we expect the ECU embedded in the vehicle to have at least a 1 GB memory.
Overall, these benchmarks show that the resource consumption, in terms of memory occupation, execution time and bandwidth, is acceptable in the context of an ECU embedded in a vehicle.

9 Discussions and Future works

From a theoretical viewpoint, this solution provides a sufficiently generic approach to handle the main challenges raised by the integration of the legislation in connected vehicles. However, several rooms of improvement are required to maintain the effectiveness of the approach in the following years. Three main challenges still need to be tackled:

1. the efficient integration of post-quantum compliance;
2. the vehicles' anonymity;
3. the inference attacks through queries.

For the first point, existing post-quantum ABE schemes are sufficiently expressive to construct the access trees required for the legislation enforcement. Nevertheless, they still require theoretical improvements before their adoption in practice. As presented in Section 4, to achieve adaptive security, constraints must be applied to the access tree. For instance, construction from Tsabary et al. [47] requires access trees to be represented in Conjunctive Normal Form (CNF), which differs from the natural derivation of access trees from the law. The transformation to CNF is possible, but increases the span of the tree, impacting both ciphertext and key sizes, and computation times. The same discussion stands for concurrent post-quantum ABE schemes. Another point is the lack of large universe constructions. Basically, to support the access of uploaded data by the vehicle itself, the identity is encoded as an attribute. Current post-quantum ABE schemes do not scale well when the universe of attributes is large. Once again, such limitation impacts both ciphertext and key sizes, and computation times. The protocol remains fully compatible with pairing-based ABE schemes, which are considered mature but are vulnerable to a quantum adversary.
For the second point, the anonymity of the vehicle cannot be guaranteed because the identity of the vehicle is included in the list of attributes. In the ABE cryptographic scheme currently used in the protocol, the attributes are sent in plaintext and as a consequence, the identity of the vehicle that sends data is not confidential. Some other ABE cryptographic schemes enable to hide the attributes (these schemes are so-called hidden-policy) but they may not necessarily be post-quantum compliant and we still have to investigate these issues.
For the last point, in the current version of the protocol, the inference attacks that may be possible through the observation of the different queries are not considered. Such attacks are still possible and would allow the storage center and possible attackers observing the communications to infer some information about the vehicle and, as a consequence, to the driver himself. Some suited countermeasures are still to be investigated to overcome these issues.
A last future work can be considered to improve the protocol performances. When a data needs to be sent by the vehicle, the S key K in the step 3 of the Secure vehicle data send scenario (Subsection 5.3.2) can be reused as a short-lived session key. This improvement would reduce the time between two emissions of messages, nevertheless, as previously stated in the Performance Evaluation section (Section 8), without this optimization the performances of the protocol are still acceptable.

10 Conclusion

In this article, a security protocol allowing to implement fine-grained access control mechanisms on the data emitted by connected vehicles is presented. This protocol mainly relies on an ABE cryptographic scheme, satisfies suited security properties formally proven with the ProVerif tool, and is designed to be post-quantum compliant. Furthermore, this protocol takes into account the legislation to generate the adequate access trees for the different stakeholders and the attributes used during data encryption. For that purpose, this article also describes a semi-automated process based on the OrBAC formalism and a specific use case illustrating the

process. Finally, the performances of the protocol have been evaluated in terms of computation time, memory consumed, and message size, and show that the performances of the protocol are acceptable in a connected vehicle context. Future work is directly connected to the challenges identified in Section 9. We plan to investigate the size of the attribute universe and identify, if necessary, a suited compromise to satisfy current post-quantum constraints. Finally, we also plan to study solutions to grant anonymity to vehicles and to prevent the possible inference attacks.

References

1. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 281–285, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
3. Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. Yapa: A generic tool for computing intruder knowledge. In Ralf Treinen, editor, *Rewriting Techniques and Applications*, pages 148–163, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
4. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy*, pages 321–334, 2007.
5. Bruno Blanchet. Symbolic and computational mechanized verification of the arinc823 avionic protocols. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 68–82, 2017.
6. Bruno Blanchet and Avik Chaudhuri. Automated formal analysis of a protocol for secure file sharing on untrusted storage. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (S&P'08)*, pages 417–431. IEEE, 2008.
7. Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2018. Originally appeared as Bruno Blanchet and Ben Smyth (2011) ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.
8. Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko, and Frederic Oehl. Improvements on the genet and klay technique to automatically verify security protocols. In *Proc. AVIS*, volume 4, page 84, 2004.
9. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenkov, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.
10. Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.
11. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International conference on the theory and applications of cryptographic techniques*, pages 453–474. Springer, 2001.
12. David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, 1991.
13. Liqun Chen and Mark Ryan. Attack, solution and verification for shared authorisation data in tcg tpm. In Pierpaolo Degano and Joshua D. Guttman, editors, *Formal Aspects in Security and Trust*, pages 201–216, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
14. Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, aes algorithm submission, september 3, 1999. *URL http://www. nist. gov/CryptoToolKit*, pages 37–38, 1999.
15. H. Deng, Q. Wu, B. Qin, J. Domingo-Ferrer, L. Zhang, J. Liu, and W. Shi. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Inf. Sci.*, pages 270–384, 2014.
16. Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.
17. Josep Domingo-Ferrer, Oriol Farras, Jordi Ribes-González, and David Sánchez. Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges. *Computer Communications*, 140:38–60, 2019.
18. M.F. Ezerman, H.T. Lee, S. Ling, K. Nguyen, and H. Wang. A provably secure group signature scheme from code-based assumptions. In *ASIACRYPT*, 2015.
19. Chaosheng Feng, Keping Yu, Moayad Aloqaily, Mamoun Alazab, Zhihan Lv, and Shahid Mumtaz. Attribute-based encryption with parallel outsourced decryption for edge intelligent iov. *IEEE Transactions on Vehicular Technology*, 69(11):13784–13795, 2020.
20. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.
21. S.D. Gordon, J. Katz, and V. Vaikuntanathan. A group signature scheme from lattice assumptions. In *ASIACRYPT*, 2010.
22. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, pages 89–98, 2006.
23. Shi-Jinn Horng, Cheng-Chung Lu, and Wanlei Zhou. An identity-based and revocable data-sharing scheme in vanets. *IEEE Transactions on Vehicular Technology*, 69(12):15933–15946, 2020.
24. Dijiang Huang and Mayank Verma. Aspe: Attribute-based secure policy enforcement in vehicular ad hoc networks. *Ad Hoc Networks*, 7(8):1526–1535, 2009.
25. Qinlong Huang, Nan Li, Zhicheng Zhang, and Yixian Yang. Secure and privacy-preserving warning message dissemination in cloud-assisted internet of vehicles. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–8. IEEE, 2019.
26. Qinlong Huang, Yixian Yang, and Yuxiang Shi. Smartveh: Secure and efficient message access control and authentication for vehicular cloud computing. *Sensors*, 18(2):666, 2018.
27. ISO. Road vehicles — controller area network (can) — part 2: High-speed medium access unit. https://www.iso.org/standard/33423.html.
28. ITU. About mobile technology and imt-2000. https://www.itu.int/osg/spu/imt-2000/technology.html#Cellular%20Standards%20for%20the%20Third%20Generation.

29. Anas Abou El Kalam, R El Baida, Philippe Balbiani, Salem Benferhat, Frédéric Cuppens, Yves Deswarte, Alexandre Miege, Claire Saurel, and Gilles Trouessin. Organization based access control. In *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 120–131. IEEE, 2003.
30. Steve Kremer and Mark D. Ryan. Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. *Electronic Notes in Theoretical Computer Science*, 128(5):87–104, 2005. Proceedings of the 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems (SecCo 2004).
31. F. Laguillaumie, A. Langlois, B. Libert, and Stehlé. Lattice-based group signatures with logarithmic signature size. In *ASIACRYPT*, 2013.
32. A. Langlois, S. Ling, K. Nguyen, and H. Wang. Lattice-based group signature scheme with verifier-local revocation. In *PKC*, 2014.
33. S. Ling, K. Nguyen, and H. Wang. Group signatures from lattices: simpler, tighter, shorter, ring-based. In *PKC*, 2015.
34. Xuejiao Liu, Yingjie Xia, Wenzhi Chen, Yang Xiang, Mohammad Mehedi Hassan, and Abdulhameed Alelaiwi. Semd: Secure and efficient message dissemination with policy enforcement in vanet. *Journal of Computer and System Sciences*, 82(8):1316–1328, 2016.
35. Wei Luo and Wenping Ma. Efficient and secure access control scheme in the standard model for vehicular cloud computing. *IEEE Access*, 6:40420–40428, 2018.
36. Shafieinejad M. and Nasr Esfahani N. A scalable post-quantum hash-based group signature. In *Designs, Codes and Cryptography*, 2021.
37. Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 696–701, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
38. D. Micciancio and C. Peikert. Hardness of sis and lwe with small parameters. In *CRYPTO*, 2013.
39. D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *SIAM J. Comput. 37*, 2007.
40. R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *CCS*, pages 195–203, 2007.
41. Jingwen Pan, Jie Cui, Lu Wei, Yan Xu, and Hong Zhong. Secure data sharing scheme for vanets based on edge computing. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–11, 2019.
42. C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, 2009.
43. O. Regev. On lattices, learning with errors, random linear codes, and cryptogra- phy. In *STOC*, 2005.
44. Sushmita Ruj, Amiya Nayak, and Ivan Stojmenovic. Improved access control mechanism in vehicular ad hoc networks. In *International Conference on Ad-Hoc Networks and Wireless*, pages 191–205. Springer, 2011.
45. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.
46. Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.
47. Rotem Tsabary. Fully secure attribute-based encryption for t-cnf from lwe. In *CRYPTO*, 2019.
48. Nyamsuren Vaanchig, Zhiguang Qin, and Batjargal Ragchaasuren. Constructing secure-channel free identity-based encryption with equality test for vehicle-data sharing in cloud computing. *Transactions on Emerging Telecommunications Technologies*, page e3896, 2020.
49. Geng Wang, Zhen Liu, and Dawu Gu. Ciphertext policy attribute-based encryption for circuits from lwe assumption. In *ICICS*, 2019.
50. B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.
51. B. Waters, M. Green, S. Hohenberger Waters, J. A. Akinyele, A. M. Dunn, and M. Rushanan. Openabe. https://github.com/zeutro/openabe.
52. Hu Xiong, Yingzhe Hou, Xin Huang, and Yanan Zhao. Secure message classification services through identity-based signcryption with equality test towards the internet of vehicles. *Vehicular Communications*, 26:100264, 2020.
53. E. Zavattoni, L. J. D. Perez, S. Mitsunari, A. H. Sanchez-Ramirez, T. Teruya, and F. Rodriguez-Henrique. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *IEEE TC*, pages 1429–1441, 2015.
54. J. Zhang and Z. Zhang. A ciphertext policy attribute-based encryption scheme without pairings. In *Inscrypt*, pages 324–340, 2011.
55. J. Zhang, Z. Zhang, and A. Ge. phertext policy attribute-based encryption from lattices. In *ASIACCS*, pages 16–17, 2012.
56. Yanan Zhao, Yingzhe Hou, Yanan Chen, Sachin Kumar, and Fuhu Deng. An efficient certificateless public key encryption with equality test toward internet of vehicles. *Transactions on Emerging Telecommunications Technologies*, page e3812, 2019.
57. Yang Zhao, Xing Zhang, Xin Xie, Yi Ding, and Sachin Kumar. A verifiable hidden policy cp-abe with decryption testing scheme and its application in vanet. *Transactions on Emerging Telecommunications Technologies*, page e3785, 2019.

A ProVerif term syntax

```
M, N ::=          terms
a,b,c,k,m,n,s     names
x,y,z             variables
(M1,...,Mk)       tuple
h(M1,...,Mk)      constructor/destructor
M = N             term equality
M <> N            term inequality
M N               conjunction
M  N              disjunction
```

B ProVerif process syntax

```
P, Q, R ::=               processes
0                         null process
P || Q                    parallel composition
!P                        replication
new n : t; P              name restriction
in(M, x : t); P           message input
out(M, N) ; P             message output
if M then P else Q        conditional
let x = M in P else Q     term evaluation
R(M1, ..., Mn)            macro usage
```

C Type declaration

```
type s_key.
type attrs.
type accessp.
type abe_mkey.
type abe_skey.
type abe_pkey.
type gs_mkey.
type gs_skey.
type gs_pkey.
```

D Global variable

```
free c: channel.
free storage_attrs: attrs.
free storage_ap: accessp.
```

E Event declaration

```
event vehicle_send(attrs, attrs, bitstring, bitstring).
event vehicle_read(attrs, bitstring, bitstring).
event vehicle_conj(accessp, attrs).
event vehicle_leak(attrs, abe_skey).
event storage_write(bitstring).
event storage_send(bitstring).
event storage_leak(abe_skey).
event stakeholder_read(accessp, bitstring, bitstring).
event stakeholder_conj(accessp, attrs).
event stakeholder_leak(accessp, abe_skey).
```

F ABE representation

```
free abe_mk: abe_mkey [private].

fun abe_pkgen(abe_mkey): abe_pkey.
fun abe_skgen(accessp, abe_mkey): abe_skey.
fun abe_enc(bitstring, attrs, abe_pkey): bitstring.
fun abe_bolt(abe_skey, attrs, abe_mkey): bitstring.
fun ext_attrs(attrs, attrs): attrs.

reduc forall m: bitstring, mk: abe_mkey, i: attrs ;
  abe_attrs(abe_enc(m, i, abe_pkgen(mk))) = i.

reduc forall m: bitstring, mk: abe_mkey, i: attrs,
    sk: abe_skey, ap: accessp ;
  abe_dec(abe_enc(m, i, abe_pkgen(mk)),
          abe_skgen(ap, mk),
          abe_bolt(abe_skgen(ap, mk), i, mk)) = m.
```

G S representation

```
fun s_enc(bitstring, s_key): bitstring.

reduc forall m: bitstring, k: s_key ;
    s_dec(s_enc(m, k), k) = m.
```

H GS representation

```
free gs_mk: gs_mkey [private].

fun gs_pkgen(gs_mkey): gs_pkey.
fun gs_skgen(attrs, gs_mkey): gs_skey.
fun gs_sign(bitstring, gs_skey): bitstring.

reduc forall m: bitstring, mk: gs_mkey, i: attrs ;
  gs_msg(gs_sign(m, gs_skgen(i, mk)), gs_pkgen(mk)) = m.
```

I Global tables

```
table list_conjs(attrs).
table list_msg(bitstring).
table list_stakeholders_sk(accessp, abe_skey).
table list_vehicles_sk(accessp, attrs, abe_skey).
table list_bolts(accessp, attrs, bitstring).
```

J Attribute creation process

```
let create_list_conjs() =
  !( new a: attrs ;
     insert list_conjs(a) ;
     out(c, a)).
```

K Vehicle process

```
let handle_vehicle(va: attrs, vap: accessp,
      vehicle_abe_sk: abe_skey, vehicle_abe_pk: abe_pkey,
      vehicle_gs_sk: gs_skey,   vehicle_gs_pk: gs_pkey) =
  !(
    (* Send a store request. *)
    new k: s_key ;
    new vehicle_nonce: bitstring ;
    let ct1 = abe_enc((s_key2bs(k), vehicle_nonce),
                      storage_attrs, vehicle_abe_pk) in
    out(c, gs_sign(ct1, vehicle_gs_sk)) ;
    (* Read the nonce to use. *)
    in(c, response: bitstring) ;
    let (storage_nonce: bitstring, =vehicle_nonce)
        = s_dec(gs_msg(response, vehicle_gs_pk), k) in
    new msg: bitstring ;
    (* Choose a conjunction to cipher the message msg. *)
    get list_bolts(=vap, conj, b) in
    let ct2 = abe_enc(msg, conj, vehicle_abe_pk) in
    let ct3 = gs_sign(s_enc((ct2, storage_nonce), k),
                      vehicle_gs_sk) in
    event vehicle_send(va, conj, ct2, msg) ;
    out(c, ct3)
  ) | !(
    (* Read a message from the storage. *)
    in(c, ct1: bitstring) ;
    let ct2 = gs_msg(ct1, vehicle_gs_pk) in
    let conj = abe_attrs(ct2) in
    get list_bolts(=vap, =conj, b) in
    let msg = abe_dec(ct2, vehicle_abe_sk, b) in
    event vehicle_read(va, ct2, msg)
  ).
```

L Storage Center process

```
let handle_storage(
      storage_abe_sk: abe_skey,
      storage_gs_sk: gs_skey,
      storage_gs_pk: gs_pkey) =
  !(
    (* Read a store request. *)
    in(c, store_request: bitstring) ;
    get list_bolts(=storage_ap, =storage_attrs, b) in
    let ct4 = abe_dec(gs_msg(store_request, storage_gs_pk),
                      storage_abe_sk, b) in
    let (s_key2bs(k), vehicle_nonce: bitstring) = ct4 in
    (* Generate and send the nonce. *)
```

```
    new storage_nonce: bitstring ;
    out(c, gs_sign(s_enc((storage_nonce, vehicle_nonce), k),
                   storage_gs_sk)) ;
    (* Read the data sent, check the nonce and
       store the message. *)
    in(c, ct3: bitstring) ;
    let (ct2: bitstring, =storage_nonce)
        = s_dec(gs_msg(ct3, storage_gs_pk), k) in
    event storage_write(ct2) ;
    insert list_msg(ct2)
  ) | !(
    (* Read and send a value. *)
    get list_msg(ct1) in
    let ct2 = gs_sign(ct1, storage_gs_sk) in
    event storage_send(ct2) ;
    out(c, ct2)
  ).
```

M Stakeholder process

```
let handle_stakeholder(
      gs_pk: gs_pkey) =
  !(
    in(c, ct1: bitstring) ;
    (* We could have the sk in parameter but we
       proceed this way to make leak easier. *)
    get list_stakeholders_sk(sap, stakeholder_a_sk) in
    let ct2 = gs_msg(ct1, gs_pk) in
    let conj = abe_attrs(ct2) in
    get list_bolts(=sap, =conj, b) in
    let msg = abe_dec(ct2, stakeholder_a_sk, b) in
    event stakeholder_read(sap, ct2, msg)
  ).
```

N Vehicle, Storage Center and Stakeholder deployment process

```
let create_storage() =
  let storage_abe_sk = abe_skgen(storage_ap, abe_mk) in
  let storage_gs_sk = gs_skgen(storage_attrs, gs_mk) in
  insert list_bolts(storage_ap, storage_attrs,
                    abe_bolt(storage_abe_sk, storage_attrs,
                             abe_mk)) ;
  phase 4 ;
  handle_storage(storage_abe_sk, storage_gs_sk,
                 gs_pkgen(gs_mk)).

let create_vehicles() =
  !(
    new va: attrs ;
    new vap: accessp ;
    let vehicle_abe_sk = abe_skgen(vap, abe_mk) in
    insert list_vehicles_sk(vap, va, vehicle_abe_sk) ;
    let vehicle_gs_sk = gs_skgen(va, gs_mk) in
    (
      phase 2 ;
      !(
        get list_conjs(a) in
        let conj = ext_attrs(a, va) in
        event vehicle_conj(vap, a) ;
        insert list_bolts(vap, conj, abe_bolt(vehicle_abe_sk,
                                              conj, abe_mk))
      )
    ) | (
      phase 4 ;
      handle_vehicle(va, vap, vehicle_abe_sk,
                     abe_pkgen(abe_mk), vehicle_gs_sk,
                     gs_pkgen(gs_mk))
    )
  ).

let create_stakeholders() =
  !(
    new sap: accessp ;
    let stakeholder_abe_sk = abe_skgen(sap, abe_mk) in
    insert list_stakeholders_sk(sap, stakeholder_abe_sk) ;
    (
      phase 2 ;
      !(
        get list_conjs(a) in
        get list_vehicles_sk(vap_private,
                             va_private, unused) in
        let conj = ext_attrs(a, va_private) in
        event stakeholder_conj(sap, a) ;
        insert list_bolts(sap, conj,
                          abe_bolt(stakeholder_abe_sk, conj,
                                   abe_mk))
      )
    ) | (
      phase 4 ;
      handle_stakeholder(gs_pkgen(gs_mk))
    ).
```

O Vehicle, Storage Center and Stakeholder ABE decryption key leak process

```
let do_vehicle_leak() =
<##ifdef VEHICLE_LEAK>
  get list_vehicles_sk(leak_vehicle_ap, leak_vehicle_attrs,
                       leak_vehicle_abe_sk) in
  event vehicle_leak(leak_vehicle_attrs,
                     leak_vehicle_abe_sk) ;
  out(c, (leak_vehicle_attrs, leak_vehicle_abe_sk)) ;
  !(
    get list_conjs(leak_a) in
    let leak_conj = ext_attrs(leak_a, leak_vehicle_attrs) in
    get list_bolts(=leak_vehicle_ap, =leak_conj, leak_b) in
    out(c, leak_b)
  ).
<##else>
  0.
<##endif>

let do_storage_leak() =
<##ifdef STORAGE_LEAK>
  let leak_storage_abe_sk = abe_skgen(storage_ap, abe_mk) in
  event storage_leak(leak_storage_abe_sk) ;
  out(c, leak_storage_abe_sk) ;
<##endif>
  0.

let do_stakeholder_leak() =
<##ifdef STAKEHOLDER_LEAK>
  get list_stakeholders_sk(leak_stakeholder_ap,
                           leak_stakeholder_abe_sk) in
  event stakeholder_leak(leak_stakeholder_ap,
                         leak_stakeholder_abe_sk) ;
  out(c, leak_stakeholder_abe_sk) ;
  !(
    get list_bolts(=leak_stakeholder_ap, leak_conj,
                   leak_b) in
    out(c, leak_b)
  ).
<##else>
  0.
<##endif>

let do_public_leak() =
  out(c, storage_attrs) ;
  out(c, abe_pkgen(abe_mk)) ;
  out(c, gs_pkgen(gs_mk)) ;
  0.

let leaks() =
  phase 3 ;
  (
    do_vehicle_leak()
    | do_storage_leak()
    | do_stakeholder_leak()
    | do_public_leak()
  ).
```

P Main process

```
process
  create_list_conjs()
  | create_storage()
  | create_vehicles()
  | create_stakeholders()
  | leaks()
```