TÉCNICO LISBOA

A Public Key Infrastructure for Securing Vehicle-to-Everything Communication

Leonardo Gonçalves

Thesis to obtain the Master of Science Degree in

Computer Engineering

Supervisor(s):   Prof. Nuno Santos
Eng. Carlos Cardoso

Examination Committee

Chairperson: Prof. Full Name
Supervisor: Prof. Prof. Nuno Santos
Member of the Committee: Prof. Full Name 3

May 2019

Dedicated to someone special...

Acknowledgments

I would like to my thank supervisors, professor Nuno Santos and Carlos Cardoso and other Multicert staff who guided and supported me through this thesis. Orientation was a key point in this work as it allowed me to keep focused and on the right path.

I would like to thank my family, who throughout my academic journey encouraged me to never stop and reach for the end. For all the patience, financial and emotional support I am grateful.

I would like to thank my closest friends, who accompany me in this project from the beginning, for standing at my side in the moments of pressure. For your help and caring I thank you all.

I would also like to acknowledge Multicert and my university for making this project possible, for the opportunity of working in a real development team and providing me with excellent means to learn and grow as a professional and person.

Resumo

A *comunicação entre veículos e coisas* (V2X) tem vindo a tornar-se um tópico de investigação e estandardização na Europa, América e Ásia. Este termo refere-se à troca de informação entre veículos e qualquer outra entidade que os possa afetar. For exemplo, outros veículos, pedestres, semáforos, sinalização de estrada, etc. A comunicação entre veículos e coisas é considerada uma característica essencial para atingir a automatização dos veículos no futuro. Estes veículos estão previstos para melhorar a segurança rodoviária, o conforto dos passageiros e autonomia de combustíveis a partir do aumento da eficiência do trânsito. Este trabalho tem como objetivo o estudo do estado da arte das comunicações V2X de um ponto de vista da ciber-segurança, assim como o desenvolvimento de um ambiente V2X. Toda a pesquisa e desenvolvimento apresentado nesa tese foi alcançado na empresa Multicert, onde foi proposto o tema, mostrando assim a sua relevância em contexto empresarial. Vamos começar por analisar soluções existentes ao nível da *Infraestrutura de Chaves Públicas* (ICP) e os esforços de estandardização necessários para suportar um ambiente V2X. Por fim, apresentamos o nosso sistema que funciona num contexto web e tem por base os standards e a ICP estudados.

Palavras-chave: privacidade, communicação, veículos, infraestrutura de chaves públicas, assinaturas digitais, certificados digitais.

Abstract

Vehicle-to-everything (V2X) communication has increasingly become the target of research and standardization efforts in Europe, America and Asia. This term refers to the exchanging of information between a vehicle and any entity that may affect the vehicle; such entities can be, for example, other vehicles, pedestrians or roadside units i.e. semaphores, road barriers, signs, etc. V2X communication is an essential feature of autonomous vehicles in the future. Such vehicles are envisioned to increase road safety, driver comfort, and fuel economization through traffic efficiency. This work aims to survey the state-of-the-art of the V2X communication from a cyber-security point of view and develop a V2X environment. The research and development presented in this document was perfrmed at Multicert, where the topic was proposed, showing the business relevance of the topic. We start by analyzing different existing solutions regarding the *Public Key Infrastructure* (PKI) and the standardization efforts needed to support a V2X driving environment. Then we present our V2X system which works in a web context and is baded on the studied standards and PKI.

x

Contents

List of Tables

List of Figures

Chapter 1

Introduction

According to the European commission statistics [1] in the year 2016 over 25000 people died in road accidents in Europe, furthermore it is estimated that for every death on Europe's roads there is 4 permanently disabling injuries such as damage to the brain or spinal cord. Intelligent transportation that is capable of assisting the driver and connect vehicles can reduce accidents significantly. *Intelligent Transportation Systems* (ITS) [2] are applications that allow vehicles to connect and coordinate their actions. This cooperation of vehicles is expected to increase road safety and traffic efficiency by assisting the drivers to make better decisions and advising new routes based on the traffic conditions.

One fundamental aspect of ITS is the V2X communication. Vehicles equipped with this technology are able to share data in real time with other vehicles, road infrastructure (roadside units) and pedestrians utilizing short-ranged wireless signals. Such data may be related to sender's presence on the road, or related to road events so that other vehicles affected by that specific occurrence (e.g. road obstacle) are notified. While vehicles transmit these types of data, roadside units transmit regional data such as speed limits, timing of semaphore lights or information about traffic deviation. Vehicles communicating with other vehicles, pedestrians and infrastructure on the road create a decentralized network known as *Vehicular Ad Hoc Network* (VANET) [3] [4]. This type of communication allows the developing of ITS applications that can signal various kinds of events, for example, cover forward collision warnings, emergency vehicle approaching, lane change warning/blind spot coverage, road works warning, and many more. Thus, V2X enhances the vehicle's perception of environment much beyond the driver's visual horizon and vehicle sensing capabilities.

Security becomes fundamental in VANETs, which are threatened by a range of potential attacks, such as distribution of forged messages, tracking of user vehicles and denial of service. The consequences of such threats can be extremely serious, and may range from disruption of the transportation to serious damage to public safety on the road. Our work focuses on a PKI mechanism that aims to address some of previous cyberattacks. The IEEE 1609.2 [5] and ETSI TS 103 097 standards [6] specify protocols for V2X communication security and recommend the usage of digital certificates to sign the messages, thus making the public key infrastructure essential. The basic idea is that all *ITS Stations* (ITS-S) i.e. vehicles and *Roadside Units* (RSU), which are equipped with a V2X communication unit

have to be registered with the PKI. Only with valid certificates these stations are able to send authenticated messages that will be trusted by the receiving stations. The certificates provided by the V2X PKI have to be stored in the hardware security module known as *On-Board Unit* (OBU) or *On-Board Equipment* (OBE).

Although this basic approach allows for message authentication, care must be taken in the design of the PKI as so to avoid privacy violations. Certificates used for V2X communications must not contain any information that links them to a particular vehicle or owner, e.g. a license plate number; such information would allow vehicle tracking by simply listening to the communications. However, removing all identifying information from certificates i.e. using pseudonym certificates is not sufficient. If a vehicle uses a single pseudonym during its lifetime, then this certificate can again be used to track the vehicle. To defeat this scheme, an attacker would only need to observe a vehicle using the same certificate at different locations to be able to link that certificate to the victim vehicle. The most common approach to assure privacy at this level is to store a pool of short-lived pseudonym certificates (also known as authorization tickets) in each vehicle's OBU. Vehicles periodically change pseudonym to authenticate V2X messages in order to avoid long-term tracking. This mechanism implies that vehicles need to communicate with the PKI to request new pseudonym certificates whenever their locally stored list is expiring. In addition to pseudonym certificates, stations also need a long-term enrollment certificate tied to their identity to authenticate within the PKI. The result is a vehicular PKI that is architecturally different from a traditional PKI.

1.1 Motivation

V2X is rapiddly developing and has been recently gaining commercial acceptance with the networking tecnologies caputing the attention of vehicle manufacturers and regulators. There are two competing V2X networking tecnologies: The original WLAN-based IEEE 802.11p/DSRC (Dedicated Short Range Communications) standard, and the more recent 3GPP-defined cellular V2X (C-V2X) technology which will eventualy evolve to 5G.

Motivated by road safety and fuel efficiency, Toyota and General Motors have already equipped some of their vehicles with 802.11p V2X in Japan and North America. Futhermore, the Volkswagen group plans to deploy the 802.11p V2X in volume models from 2019 onwards, claiming that this standard has been comprehensivly tested for vehicle communications [7].

On the other side of the spectrum, recent studies and analysis [8] [9] performed by the 5G Automotive Association (5GAA), the organization supporting and developing C-V2X tecnology, indicate that cellular-based C-V2X in direct communications is ahead of the 802.11p standard in terms of performance, range, reliability and others. In 2019 5GAA performed a live demo event in Berlin [10] demonstrating some of the V2X use cases: traffic management solutions, real time emergency alerts, live data capture and transmission, combined network and direct solution and even remote-operated driving.

There is an extensive amount of literature regarding the the WLAN versus C-V2X debate. However, we will not be going into detail on these works bcause they approach networking and other aspects of

the V2X which are behond scope of this thesis.

Regardless of the ongoing debate, regulation uncertainties, and other challenges, the spending in V2X communication technlogy is expeteced to increase more than 170% between the years of 2019 and 2022. Furthermore, this technology is expected to reduce the number of road accidents by 80%, easy traffic congestion by 20% and reduce fuel consumption and emissions by 10% to 40%. These findings are part of the are part of SNS Telecom & IT's latest research report on V2X communications technology [11], which is based on large scale pilos, projects and early commercial rollouts.

1.2 Topic Overview

The proposed solution consists of a vehicular PKI and a demo simulator to evaluate its correctness. Our solution will extend mPKI, a currently operating traditional PKI which is the product of Multicert. In order to extend mPKI, we will start by developing a Java package that implements the new certificate formats, V2X messages and certificate requests. The next step involves integrating such package in mPKI to allow it to issue the certificates for the end-entities (ITS-S) and CAs. The final step is to develop a simulator to communicate with the PKI. In this phase we will develop a Java project that will simulate the end-entities and the interaction between such end-entities (V2X) and the vehicular PKI.

1.3 Objectives

This work addresses the problem of designing and implementing a vehicular PKI solution that allows for V2X message authentication while preserving the privacy of its users. This report will specify the system to produce, a vehicular PKI that supports the enrollment of new ITS-S, provisioning of valid certificates to its users and the removal of compromised stations or PKI entities. The goal of this work is to implement a PKI solution based on the most recent European standards and follows the following requirements.

- Privacy

  - The drivers must remain anonymous on the road, meaning that unauthorized parties are not able to associate a V2X message to the vehicle/driver who sent it.

  - The messages transmitted during V2X communications must remain unlinkable to the vehicles which previously sent them.

  - The privacy of the vehicle's location should be protected by the usage of pseudonym certificates.

  - Deducing the vehicle's location or tracking vehicles should not be aided by analysing the vehicle's previous V2X communications.

- Confidentiality

  - Information transmitted to or from a given vehicle to the PKI, such as certificate requests and responses, should be protected against unauthorized access.

- Integrity

  - Information transmitted to or from a given vehicle must be protected against unauthorized modifications or tampering during transmission.

- Authenticity

  - When a vehicle receives a V2X message it should be able to trust that such message is relevant and was created by a legitimate ITS-station.

  - When a vehicle receives a message from the PKI, such as a certificate response, it should be able to trust such message was created by the intended CA as a response to the initial request.

  - Configuration information originated from the PKI should arrive to the vehicles in a state that allows them to confirm the origin and integrity of the message.

- Availability

  - Access to the PKI services should not be prevented to legitimate vehicles by malicious activity.

1.4   Thesis Outline

The remainder of this report is organized as follows: Chapter 2 provides an overview of the state-of-the-art regarding the technologies and mechanisms used to manage the identity of vehicles during V2X communications. Chapter 3 refers to the architecture and implementation of the solution to achieve the goals previously described. In this chapter we first invite the reader to understand from a high level what are the main system components, the goal of each individual component and how they work together to form the solution. Then we provide details about the implementation of such components, explaining the main technologies and mechanisms behind them. Chapter 4 provided an experimental evaluation of the implemented system. Here we display the performance tests done to the solution and describe the security properties achieved. Finally, 5 concludes this document by summarizing the work, describing the achieved goals and suggesting future work to improve the solution.

Chapter 2

State of the Art

In this section we analyze the existing work related to V2X communication. In order to have a high level understanding of how V2X communications work, we start by specifying two vehicular PKI solutions. The first is named *A Generic Public Key Infrastructure for Securing Car-to-X Communication* and has been proposed by the corresponding stakeholders in Europe [12] [6] [13] (Section 2.1). The second is named *Security Credential Management System* (SCMS) and is the American counterpart proposed in [14] (Section 2.2). For each solution, we analyze the corresponding PKI architecture and its most relevant operational aspects such as enrollment of vehicles, revocation of certificates, and others. In Section 2.3, we provide an overview the existing standards behind the European solution, which will be the basis of our work, and present a lower level and more detailed notion of the V2X communication functioning. Lastly in Section 2.4, we introduce the existing V2X communication simulators and study how they can be used to evaluate our vehicular PKI solution.

2.1 Overview of the European Vehicular PKI Solution

The European vehicular PKI solution is a generic concept so it allows some flexibility in its implementation. Our proposed PKI will primarily be based on the European solution. Specifically we aim to readjust such solution so it can be integrated in Multicert's PKI.

2.1.1 European Vehicular PKI Architecture

The European PKI uses long-term certificates named enrollment certificates and short-term certificates known as pseudonym certificates or authorization tickets. Enrollment certificates are tied to the vehicle's identity to authenticate the vehicle within the PKI back-end. Authorization tickets have the identifying information removed and are used in V2X communications for privacy reasons. The European PKI considers an hierarchical structure as we can see in Figure 2.1. Such architecture is composed of a *Root Certification Authority* (RCA), an *Enrollment Authority* (EA), and an *Authorization Authority* (AA). For a given trust domain the RCA certificate is the root of trust for all certificates in that hierarchy, this means that a vehicle will only trust an incoming message if the certification chain starting on the received
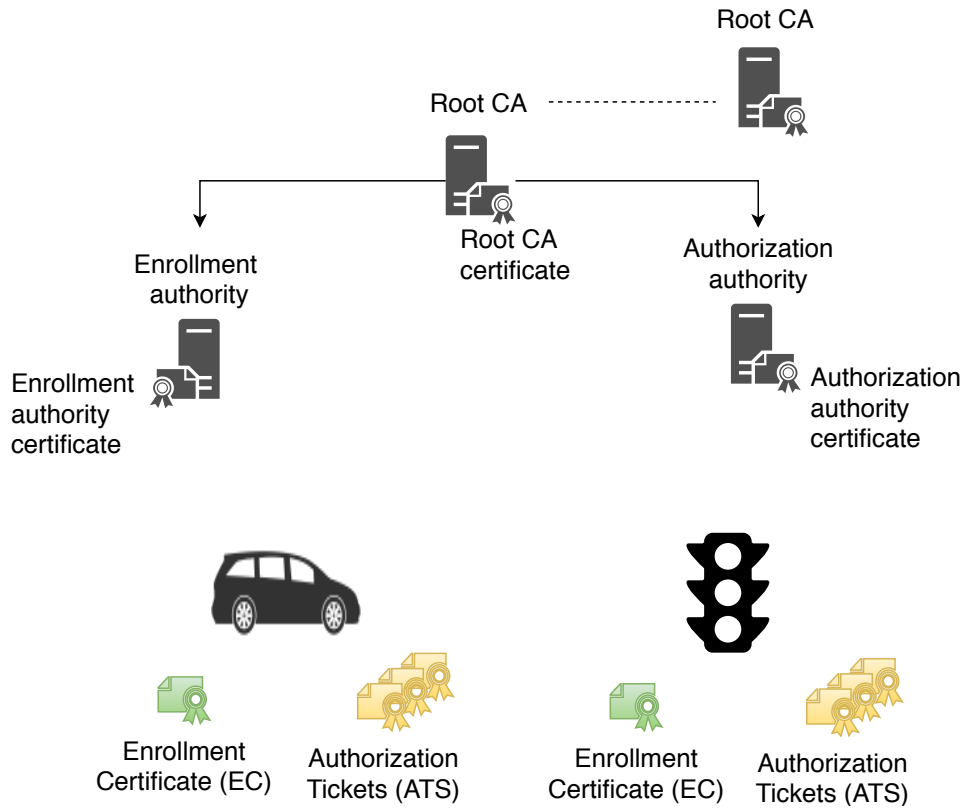
Figure 2.1: European vehicular PKI architecture.

authorization ticket to the root CA certificate is valid. The RCA is responsible for issuing certificates for enrollment authorities and authorization authorities. If there are multiple RCAs, trust between them can be established by using cross certification. No other cross certification between CAs is allowed. The EA has the responsibility of validating that a vehicle can be trusted and only if so, issuing an enrollment certificate for that vehicle as a proof of its identity. Finally, the AA exists to allow vehicles to apply for specific services and permissions on the road. These privileges are denoted by means of authorization tickets (pseudonyms), which are issued by the AA for the applying ITS-S.

2.1.2 ITS-S Security Life Cycle

The ITS-S security life cycle is relevant for our solution because through its analysis we are able to understand which stages every vehicle undergoes during its life time. Each stage involves a change in the vehicle's state against the PKI. The analysis of such stages allows us to understand when and what information needs to be initialized in the vehicles and transfered between vehicles and the PKI. Specifically, which variables need to be initialized in the vehicles' OBU at bootstrap, what needs to be done in regards to the vehicle enrollment and authorization, and how can we update PKI management information throughout the life cycle of the vehicles. ETSI TS 102 941 [15] standards specify the ITS-S' security life cycle comprising four stages: the manufacture, enrollment, authorization and maintenance. The first stage of an ITS-S' life cycle is the manufacture, it is at this stage that all the information needed for the enrollment is initialized within the station itself (OBU) and within the EA. The next stage is the en-

rollment where the ITS-S requests its enrollment certificate from the EA. Having received the enrollment certificate, the ITS-S can now request authorization tickets from the AA. The request for authorization tickets represents the authorization stage.

When a vehicle is manufactured the OBU and EA need to be initialized. Within the vehicle's OBU, it is necessary to provide information regarding the vehicle's identity and information to allow the vehicle to interact with the PKI. In regards to identifying information, an unique identifier and a public/private key pair to be used for cryptography are created. Optionally, a canonical certificate can be installed which associates the canonical identifier with the public key of the vehicle. In this case, the certificate chain back the root authority needs also to be installed. To allow a vehicle to connect to the PKI, it is necessary to install the network address and public key certificate of the EA and AA that will issue certificates for that vehicle. In addition, the set of known and trusted EA certificates is installed to allow the vehicle to initiate the enrollment process. To grant that such vehicle is able to verify the authenticity of incoming V2X messages, the set of known and trusted AA certificates must also be provided. In order to support vehicle enrollment, within the EA it is necessary to provide information that identifies the manufactured vehicle: a unique vehicle identifier, the location profile information for the vehicle, and the public key that belongs to the vehicle's key pair. In the next sections we study how the European Vehicular PKI operates regarding the provisioning of certificates, their revocation, and how the messages are signed and verified by the user vehicles. Our vehicular PKI must support such operational aspects.

2.1.3 Enrollment Process

Before an ITS-S is able to participate in the V2X communication it must be registered within the PKI. The enrollment request message shall be sent from the ITS-S to the Enrollment Authority, to protect the users privacy the request must be encrypted. According to ETSI TS 102 941 [15] this message should contain the following fields:

- Message signer information, i.e. the canonical certificate or the public key provided to the ITS-S at bootstrap to globally identify it.

- certificate request, i.e. the information to be presented in the enrollment certificate. For example, the ITS-S' public key, start time, end time and other certificate specific data.

- The digital signature of the message sender (requesting ITS-S) calculated over all of the message fields.

After the ITS-S enrollment request the target EA must reply with a successful or failed response message, to protect user privacy the response shall also be encrypted. The successful ITS-S enrollment response should contain the enrollment certificate and the chain of certificates back to the originating enrollment CA. In the failed ITS-S enrollment, the response contains the error code i.e. the reason for the unsuccessful enrollment response.

2.1.4 Authorization Ticket Provisioning

Pseudonym certificates are short-lived certificates which express the permissions that a specific enrolled vehicle has on the road while hiding its identity. Consequently they are refereed as *Authorization Tickets* (ATs) by ETSI. To avoid long-term tracking, a vehicle rotates authentication tickets from its local pool to authenticate V2X messages. However, it needs to request new ATs from the authorization authority once there are few valid ATs stored locally. The update can be done over-the-air or at an authorized dealership (during vehicle maintenance), i.e. roadside-units and workshops can act as a proxy for certificate requests. Because our solution will also use ATs, important decisions must be done regarding the frequency that ATs are provisioned to the enrolled vehicles and how such vehicles rotate certificates from their pool. There is the need to adopt a model that specifies the authorization ticket provisioning. The model used in Europe is defined in CAR 2 CAR Communication Consortium (C2C-CC) [12]. In this model the required frequency of updates, the delivered level of privacy and security can be expressed by three determining parameters:

- **Certificates valid simultaneously:** It can be defined that the requester can use several ATs with the same start and expiry date.

- **Authorization ticket validity time period:** Is defined by the time between start and expiry timestamps of the ATs.

- **Overall covered time-span:** The time that is covered by the batch of authorization tickets.

A vehicle receives a "super-batch" that contains a set of ATs, the duration of the "super-batch" is the *overall covered time-span* and is in the order of years. The "super-batch" is composed of "sub-batches" which contain a sub-set of certificates (e.g., 20) valid for the same time period (e.g., a week). During that week the vehicle uses and reuses the 20 valid certificates. The certificate usage pattern can vary from device to device, e.g. a device could use a certificate for 5 minutes after start-up, then switch to another certificate, and use that either for 5 minutes, or until the end of the journey. Figure 2.2 illustrates this method.

2.1.5 Authorization Ticket Request Process

In this section we study the sequence of messages used by vehicles to request valid ATs in Europe. The solution that we propose will assume such protocol for the simulated vehicles' requests for ATs. At a high level, a vehicle uses its enrollment certificate to prove its enrollment to the AA, only then the AA can issue the ATs. The ETSI TS 102 941 [15] standard specifies in detail the message format for the AT request and response. In regards to the process [12], the vehicle sends a request to a predefined AA. The request includes the vehicle's enrollment certificate, the certificate of the corresponding Enrollment Authority, and the list of public keys. To protect user's privacy the enrollment certificate may be encrypted with the public key of the corresponding EA. In this case the AA is not able to create a link between the authorization tickets and the enrollment certificate of a specific vehicle. Consequently, when an AA
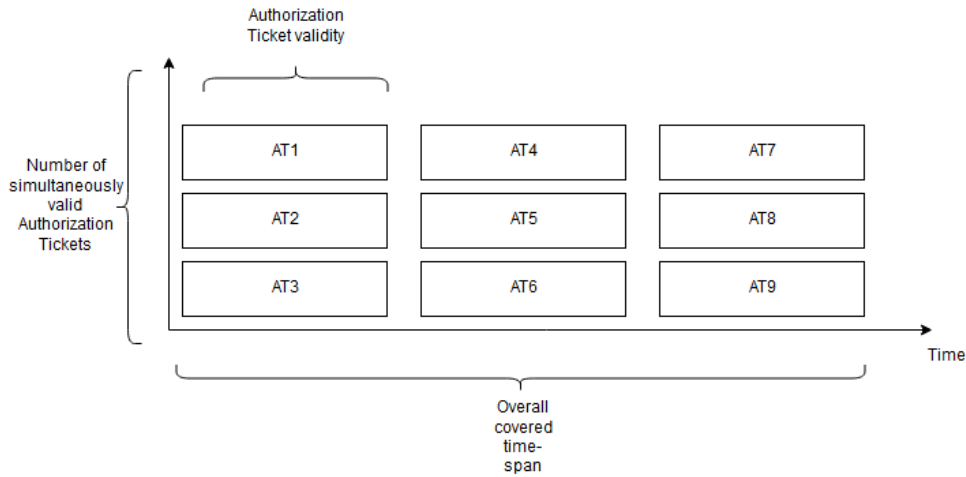
Figure 2.2: Authorization Ticket provisioning model.

receives such requests it cannot verify the enrollment of the requesting vehicle. In order to do so, the AA sends a request with the (encrypted) vehicle's enrollment certificate and the calculated AT overall covered time-span (e.g. 1 year) to the correct EA (identified by the EA certificate present in the original request). The EA maintains a database that stores a timestamp marking the deadline which the vehicle will still have valid ATs (calculated using the overall covered time-span of the request). Only if the vehicle's enrollment certificate is valid and no ATs are issued for the time which the vehicle still has valid ATs the AA will get a positive response from the EA validating the enrollment of the vehicle. Upon receiving such response, the AA has the responsibility of issuing the ATs for the vehicle. This procedure prevents a vehicle from requesting ATs for the same time interval from different or the same AAs.

2.1.6 Message Signing and Verification

In this section we analyze how the secured V2X messages are signed and verified by the vehicles. Such information will allow us to correctly test the communications between vehicles in the proposed simulator. In regards to sending messages, the sender of V2X messages signs all outgoing messages with the private key of a valid AT. Afterwards, the message with the appended signature and pseudonym certificate is broadcast. When a station receives a message, the senders authenticity and message integrity is verified by decrypting the signature with the public key from the appended AT. The sender's authenticity is only accepted if verification of the received AT up to a root CA is possible. Vehicles are preloaded with the known and trusted authorization authority certificates at manufacture. However if the Authorization Authority certificate that corresponds to the received AT is not locally stored, the message receiver cannot validate the sender's authenticity. In this case, the message receiver must create a new message requesting the missing Authorization Authority certificate and send it to the original message sender. Then, the receiver of this request must respond with the Authorization Authority certificate (more details present in Section 2.3).

2.1.7 Certificate Revocation

Sometimes it may be necessary to remove bad actors from the system. This requirement influences the architecture of the PKI, namely there must be an entity responsible for detecting misbehaving actors. Once detected, a bad actor must be removed from the communication.

In the European solution detecting and preventing misbehavior by means of a misbehavior entity is not yet supported. Revocation is done in respect to the long-term enrollment certificate of ITS-S and CA certificates. The ITS stations are eventually removed from the system by rejecting new requests for ATs. In this concept the EA links the revocation information of the vehicle to its long-term enrollment certificate. If the vehicle requests new ATs then the AA forwards the request to the respective EA which checks the revocation information of the requester. In respect to the revocation of any CA certificate a distributed CRL is used. In this scheme, the CA certificates that are compromised are revoked manually by the PKI administration; the certificate identifier is posted in the CRL and signed by the root CA; finally, the CRL is distributed inside the PKI backbone and connected ITS stations. The CRL for EA and AA certificates is defined in ETSI TS 102 941 [15] standard

2.2 Overview of the American Vehicular PKI Solution

In this section we provide an overview of the Security Credential Management System as a matter of reference. Although this theses follows the European PKI solution, it is relevant to reference the American vehicular PKI in order to understand the main differences and similarities between solutions. In comparison to the architecture of the European vehicular PKI the American counterpart is noticeably more complex. However, it shares some similarities with the European vehicular PKI. The main differences include an increased focus on privacy against attacks from SCMS insiders, the handling of certificate revocation, and the method for provisioning certificates based on the butterflykey expansion algorithm.

2.2.1 American Vehicular PKI Architecture

SCMS considers an hierarchical structure as we can see in Figure 2.3. Comparing with the European vehicular PKI architecture, there are some components with a similar function and others that introduce new functionality. In regards to the similar components, SCMS assumes a *Root CA*, an *Enrollment CA*, and a *Pseudonym CA* (PCA) which corresponds to the authorization authority in the European architecture. As to the remaining components, their functionality is as follows:

- **Device Configuration Manager (DCM):** Provides authenticated information about changes in the configuration of SCMS's components, for example an authority changing its certificate. It is also used to inform an enrollment CA that a device is eligible to receive enrollment certificates.

- **Registration Authority (RA):** Validates, processes, and forwards requests for pseudonym certificates to a pseudonym CA.

- **CRL Store (CRLS):** Stores and distributes CRLs. CRLs are signed by the CRL Generator.

- **CRL Broadcast (CRLB):** Broadcasts the current CRL, may be done through road side units or satellite radio system, etc.

- **Linkage Authority (LA):** The main goal of this component is to improve certificate revocation. LAs generate linkage values, which are used in the certificates and support efficient revocation (more on this later). There are two LAs in the SCMS, referred to as LA1 and LA2. The splitting prevents the operator of an LA from linking certificates belonging to a particular device.

- **Location Obscurer Proxy (LOP):** The main goal of this component is to improve the security against SCMS insiders. The LOP hides the location of the requesting device by changing source addresses. Additionally, when forwarding information to the Misbehavior Authority (MA), the LOP shuffles the reports to prevent the MA from determining the reporters' routes.

- **Misbehavior Authority (MA):** Processes misbehavior reports to identify potential misbehavior by devices, and if necessary revokes and adds devices to the CRL. It also initiates the process of linking a pseudonym certificate to the corresponding enrollment certificates, and adding the enrollment certificate to an internal blacklist.

- **Request Coordination (RC):** Ensures that a device does not request more than one set of pseudonyms for a given time period. It coordinates activities between different RAs.

An early version of SCMS has already been implemented, operated and tested in the safety pilot project [16]. Safety pilot is a scaled-down evaluation of V2X that uses real vehicles and roadside infrastructure in order to understand the safety benefits of connecting vehicles. SCMS documentation can be found in [17]

2.2.2 Pseudonym Certificate Provisioning Model

Regarding the model used for provisioning pseudonym certificates, the SCMS assumes the same model as in CAR 2 CAR Communication Consortium [12] (European solution) illustrated in Figure 2.2. The proposed parameters for this model are:

- **Certificate validity time period:** 1 week.

- **Certificates valid simultaneously (batch size)**: 20 to 40 certificates.

- **Overall covered time-span (super-batch size):** 1 to 3 years.

2.2.3 Pseudonym Certificate Request Process

The request for pseudonym certificates in itself is different from the European solution. For this the butterfly key expansion algorithm [14] is used. Butterfly keys allow a device to request an arbitrary number of certificates, each encrypted with a different encryption key and each containing a different signing key. The request contains only one seed for the verification public key, one seed for the encryption public key, and two expansion functions. Without butterfly keys, vehicles would have to send a signing key
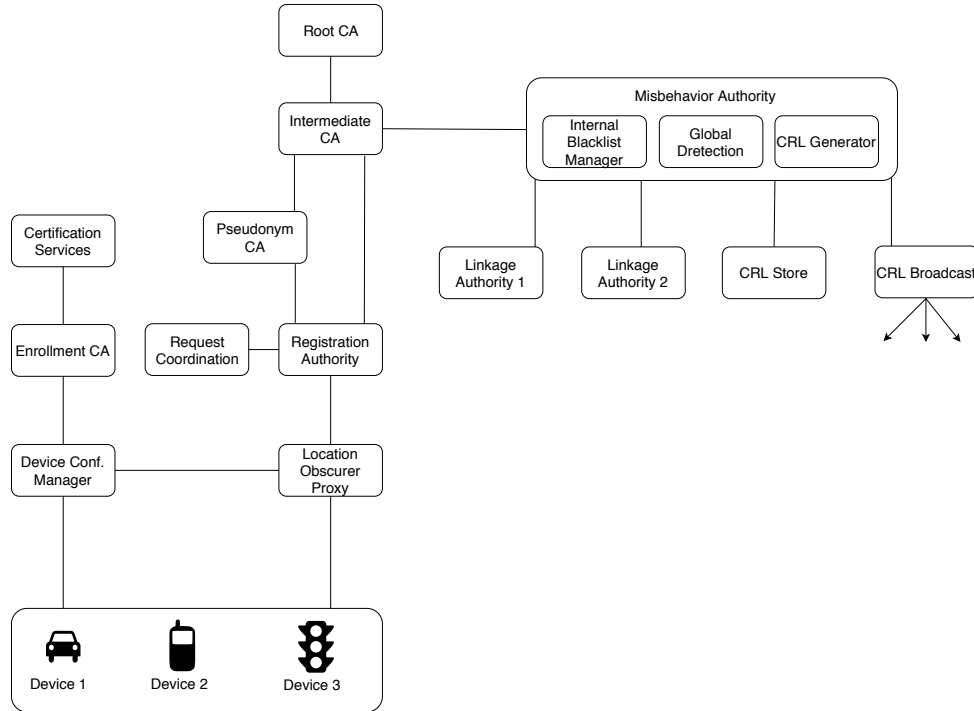
Figure 2.3: SCMS overall system architecture.

and a unique encryption key for each requested certificate. Butterfly keys reduce upload size, allowing requests to be made even in suboptimal connectivity conditions, and also reduce the computation to be done by the vehicle to calculate the keys. More information about the request process present in [14]:

2.2.4 Misbehavior Reporting

In contrast to the European PKI, the American PKI supports misbehavior reporting by user vehicles. This feature aims to improve the security against SCMS outsiders by reporting their malicious messages. Devices will send misbehavior reports to the MA via the LOP which will obscure the source and shuffle the reports from multiple reporters, this is done to prevent the MA from reconstructing the reporter's path based on the reports. The format of a misbehavior report is not fully defined yet, but a report will potentially include reported messages in addiction to the reporter's signature and certificate, and will be encrypted by the reporter for the MA.

2.2.5 Global Misbehavior Detecting and Revocation

The algorithms necessary for global misbehavior detection have not been developed at the time of this writing. However, the interface which allows SCMS components to retrieve linkage information is already specified. Revocation is tightly bound to the linkage information which basically allows the MA to find whether multiple reported messages point to the same device. The revocation process is described step-by-step in [14]. In this section we learned about the European and American vehicular PKIs and about their most relevant operational aspects. In the next section we present the standards which shape the formats of the certificates and messages used in the European vehicular PKI.

SignedMessage	SignedMessage
Protocol Version	Content

Certificate

Signer's certificate

───certificate , certificate chain, or digest of certificate───

HashAlgorithm	ToBeSignedData	SignerIdentifier	Signature
Hash identifier	Content to be signed	Signer identifier	Signature

ToBeSignedData

Content to be signed

Sign and hash

SignedDataPayload	HeaderInfo
Message payload	Security header

Signer's private key

Psid,	Time64	Time64	ThreeDLocation	MissingCrlIdentifier	EncryptionKey	SequenceOfHashedId3	Certificate
provider service identifier	generation time (opt.)	expiration time (opt.)	generation location (opt.)	missing CRL identifier (opt.)	encryption key (opt.)	inlineP2pcdRequest (opt)	requested certificate (opt.)

Figure 2.4: IEEE 1609.2 signed message format used by ETSI TS 103 097.

2.3 Secured Message and Certificate Formats Standard

One of the main concerns of V2X communication is the ITS interoperability. Standardization of the communication protocol becomes fundamental with so many vehicles from different manufacturers using the road and sharing information. To achieve this goal there are dedicated work groups within standardization organizations that address security and privacy concerns. While ETSI Automotive Intelligent Transport Systems represents the main standardization stakeholders in Europe [18], IEEE 1609.2 represents the main standardization stakeholders in the U.S [5]. Such standardization efforts are the basis of the security and privacy of the European and American vehicular PKI solutions respectively. A survey about recent standardization activities in Europe (ETSI) has been done by IEEE in [13].

In regards to the secured message and certificates formats. IEEE 1609.2 [5] standard defines the formats for secured V2X messages and public key certificates to be used in SCMS. In this standard the V2X message authenticity and integrity are based on the *Elliptic Curve Digital Signature Algorithm* (ECDSA). The message confidentiality protection is based on AES symmetric encryption (AES-CCM mode). For the transport of symmetric keys the *Elliptic Curve Integrated Encryption Scheme* (ECIES) is used. ETSI TS 103 097 standard [6] assumes the same cryptosystem as IEEE 1609.2 and presents security profiles for the messages and certificates also based on the IEEE 1609.2 standard. This means that ETSI's profiles are specific types of messages and certificates which are based on particular options available on the definitions of the base standard. For example, ETSI TS 103 097 uses the definition of possible fields that a certificate may contain (the format) present in IEEE and, based on these options, builds the specific profiles (the necessary fields) for the root CA certificates, authorization tickets, enrollment certificates, and other certificates to be used in the European solution. The same process applies with the profiles for the secured V2X messages.

The PKI solution that we propose will be primarily based on the European PKI. Consequently, it is relevant that we understand the contents of the secured V2X messages and the certificates used by it. In order to do so, we provide an overview of the ETSI TS 103 097 standard.

2.3.1 Secured Messages Formats

Figure 2.4 depicts the format of a secured message. Generally a message can be transmitted as unsecured data, signed, encrypted, or signed and encrypted data. In the case of road security messages, the message should be signed and thereby include the hash algorithm, the content to be signed, the signer identifier, and the signature itself. Such security messages need to carry the signing certificate (authorization ticket) to reduce the processing delay at the receiver side. However, in order to reduce the network bandwidth consumption it is possible to include in the message a reference to the signing authorization ticket. For this purpose, the message signer identifier contains the certificate identifier as the 8 bytes certificate digest instead of the full certificate.

The content to be signed includes all of the message components that will be protected by the signature. Such components are the security header and the message payload. The security header includes components that are relevant for the security layer, such as the provider service identifier and some optional message validation data (generation time, expiration time, generation location, missing CRL identifier, encryption key, *inlineP2pcdRequest,* and requested certificate).

A message is signed by an authorization ticket, which is turn is signed by an authorization authority certificate. Consecutively the authorization authority certificate is signed by a higher authorization authority certificate. The chain ends at a root CA certificate which issued itself. At a high level, at least one certificate in this chain must be known and trusted by the receiving station in order for it to be able to trust an incoming message.

The message receiver needs to be able to construct a chain from the message signing certificate to a known root. However, vehicles are constantly rotating authorization tickets to sign safety messages, normally exchanging only a reference to that certificate (authorization ticket hash). In addition, in many cases vehicles share the road with previously unknown vehicles for the first time. There needs a peer-to-peer (p2p) mechanism to distribute certificates on the road. This mechanism in embedded into the secured messages, specifically the receiver can use the *inlineP2pcdRequest* component of a message to request unknown certificates from other senders. This functionality allows us test the correctness, performance and overhead of the distribution of unknown authorization tickets on the road.

2.3.2 Secure Messages Profiles

In the previous section we learned about the possible message components and their meaning. Here, we analyze the already standardized safety message profiles for the *cooperative awareness messages* and *decentralized environmental notification messages*. For each type of message, we analyze their goal, how they are sent, and how they are received by the vehicles. The proposed simulator will implement the V2X communications between simulated vehicles. As such, it assumes these two specific types of messages secured by the ATs provided by the proposed vehicular PKI.

Security Profile for Cooperative Awareness Messages

Cooperative awareness messages (CAM) are messages that are exchanged between ITS-S. As the name implies, these messages are used to achieve cooperative awareness on the road. This means that road users such as vehicles (cars, trucks, trains, etc.), road-side units (traffic lights, gates, barriers, etc.) and people are aware of each other's positions, speed and other dynamic variables. To achieve this goal, it is essential that this type of messages is periodically broadcast by each road user to all its neighbors. CAMs are used to support traffic management and safety services. In the normal cases CAMs are sent multiple times per second with the component signer identifier containing the reference of the signing authorization ticket (8 byte certificate digest). However, in oder to distribute the currently used AT, every second a CAM is sent with the signer identifier containing the full certificate. If a vehicle receives a CAM signed by a previously unknown AT, it should include the currently used AT immediately in its next CAM, instead of including just the digest. In this case, the timer for the next inclusion of the full certificate shall be restarted to one second.

Besides distributing the currently used AT a vehicle also needs to request the unknown certificate present on the revived CAM for message verification purposes. Specifically, if a vehicle receives a CAM with the signer identifier containing an unknown certificate digest, then it will include that digest in the component *inlineP2pcdRequest* of its next CAM to broadcast the request for the full certificate. It is also possible for a vehicle to receive a CAM containing the full signing authorization ticket but this certificate is signed by an unknown authorization authority certificate. In this case the vehicle should include in the *inlineP2pcdRequest* of its next CAM the digest of the unknown authorization authority certificate which is present on the AT itself (see more in Section Certificate formats).

If a vehicle receives a CAM containing a request for an unknown certificate i.e. with the component *inlineP2pcdRequest* on the security header, then the vehicle searches the list of certificate of digests existing in that component. If the digest of the currently used authorization ticket is found in that list, then it includes the full certificate in the component signer identifier of its next CAM instead of the digest. In the case that a vehicle finds a digest referencing a valid authorization authority certificate in that list, it should include such certificate in the component requested certificate of its next CAM to broadcast the response. It is possible that multiple neighbor vehicles have stored the requested AA certificate, in order to prevent unnecessary broadcast responses, a vehicle only includes the AA certificate in its next CAM if before the generation of this message no other CAM was revived containing the AA certificate in the component requested certificate.

Security Profile for Decentralized Environmental Notification Messages

Decentralized environmental notification messages (DENM) are messages designed to provide asynchronous warning notifications to vehicles. DENMs are event triggered and are broadcast to notify the users of a hazardous event. For example, an emergency vehicle approaching or an accident on the road. These messages have to be broadcast to all users affected by the event, sometimes multiple hops are needed to achieve this.

In order to reduce the verification delay at the reviver side CAMs are always sent with the full signing authorization ticket in the signer identifier. Because it is important for vehicles to know were the event occurred, these messages will always include in the header the generation location.

2.3.3 Certificate Formats

In the previous sections we have seen the secured message formats and profiles, which are relevant for V2X communications. In this section we introduce the existing certificate formats, which are relevant to secure such messages. Our goal is to encode such formats in a Java package and then integrate it into mPKI.

The certificate formats include profiles for the root CA, enrollment authority, authorization authority and end-entities certificates (authorization tickets and enrollment certificates). Generally a certificate is composed of the issuer identifier, certificate identifier, application permissions, permitted geographic location, start of the validity time, expiration time, public key and the signature. In order to construct the certification chain, each non-root certificate carries the issuer identifier which is a reference (8 byte digest) that points to the certificate that belongs to the issuer CA. For example, authorization tickets carry the digest of their corresponding authorization authority certificate. The certificate identifier is a unique name that identifies the certificate's host (i.e. name of a certification authority in the case of CA certificates). The application permissions contain one or more pairs of provider service identifier (PSID) and service specific permissions (SSP). While PSID indicates a specific service, message or application the SSP indicates the permissions within that service. For example, there may be an SSP value associated with PSID of a CAM that indicates that the vehicle is privileged to send such message for a specific vehicle role (e.g emergency vehicle) or for a specific roadside unit (e.g traffic lights).

2.3.4 Signed Message Validity Checks

Before a vehicle is able to trust an incoming message it must check the message's validity. This is done by verifying that no certificate in the certification chain is revoked and the signing certificate chains to an already trusted CA certificate. Also the signature present in the message can be verified using the public key expressed in the certificate. The message payload must be consistent with the permissions expressed in the certificate (by the PSID/SSP pairs). Finally the message must not be not expired, i.e. the message validity is within the certificate's validity period and the message was generated within the permitted geographic location of the signing certificate. Figure 2.5 depicts the consistency between a signed message and the signing authorization ticket.

2.4 Overview of the ITS Simulators

Implementing V2X communication is expensive and may prove to be dangerous to test using real vehicles. Furthermore, to properly measure the benefits of V2X communication we need to evaluate it at
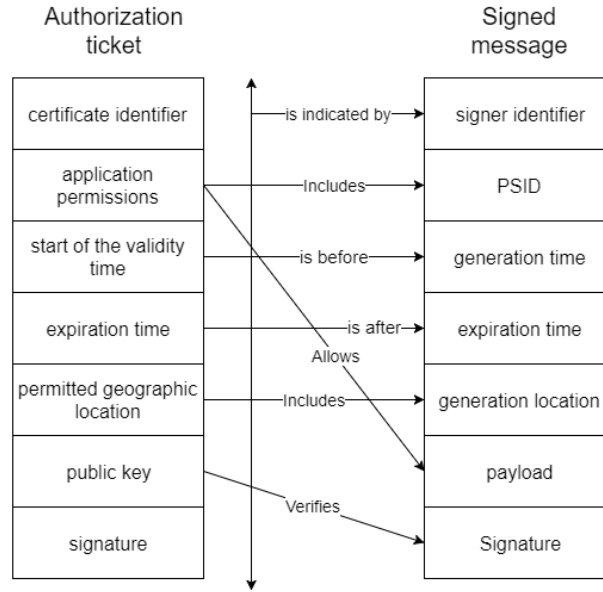
Figure 2.5: Relation between a signed message and the signing certificate.

a large scale, for example hundreds of ITS-S. Before we can conduct a field test, a simulation framework which is able to test the communications between vehicles and infrastructure of whole cities is needed. This implies simulation at three different domains: traffic simulation to generate the road networks and traffic demand; network simulation to allow vehicle connectivity by wireless technology (e.g. IEEE 802.11p DSRC, and IEEE 1609.4 WAVE) [19] [20]; and ITS application simulation to trigger the communication. In this section we list some of the most known ITS simulators. A survey on the most known simulation tools and techniques for vehicular communications and applications can be found in [21].

2.4.1 Vehicle Mobility and Networking Simulators

Vehicle mobility simulators are specialized in generating the road networks and traffic demand. At this level of simulation it is important to support: a realistic representation of traffic flow that may range from a single road junction to a whole city; the support for adding new functionality and integration with other simulation tools (e.g. an interface that allows to retrieve traffic simulation data and control the simulation using external functions). In this category of simulators there are two promising candidates: SUMO [22] and VISSIM [23]. Network simulators have the responsibility of representing the network protocols that transmit ITS information through the VANET, to a back-end or Internet service. In this category of simulators there are three promising candidates: ns-3 [24], OMNeT++ [25] and JiST/SWANS [26].

2.4.2 Integrated ITS Simulators

Integrated ITS Simulators are frameworks that couple different domain simulators in order to create a functional V2X environment. At this level of simulators it is important to support a bidirectionally-coupled simulation [27] of road traffic and network traffic (the mobility of vehicles affects communication and

vice-versa). In this category of simulators there are three promising candidates: Veins [27], iTETRIS [28], and VSimRTI [29].

2.5 Discussion

In this section we provide a brief overview of the vehicular PKI solutions presented and analyze their advantages and disadvantages. We have seen that in Europe exists *A Generic Public Key Infrastructure for Securing Car-to-X Communication* [12] and in America exists the *Security Credential Management System* [14]. In regards to the European PKI, the first disadvantage comes in the vehicle's request for authorization tickets. This solution assumes that every vehicle has to calculate a list of keypairs containing one signing and verification key for each of the requested authorization tickets. Since vehicles typically request a bundle of certificates to be used in a timespan of years, the generation of keys results in an increased computing overhead within the OBU whenever a vehicle needs to request new Authorization tickets. In addition, this process also increases the size of the request, which has to contain all of the verification keys. The second disadvantage comes in the revocation of certificates. The European PKI does not consider distribution of CRLs containing authorization tickets within the vehicular network. As a result, this solution allows a window of vulnerability where malicious vehicles have their enrollment certificate revoked but still have a pool of valid authorization tickets, which allows them to send authenticated message for the duration of that pool. Although this system has these disadvantages it provides a simple architecture that is compatible with mPKI and is based on the most accessible standards. These advantages provide us with a good starting point for the implementation of the proposed solution. In regards to the American solution, the main disadvantages are that the underlying standard is payed to obtain and most importantly, the complexity of its architecture and protocol makes it much less compatible with mPKI.

Having in mind the advantages and disadvantages of the existing vehicular PKI solutions, we decided to base our PKI solution on the European vehicular PKI. However, as we have discussed before, the European vehicular PKI is a generic concept. For this reason it cannot be immediately implemented in mPKI. For example, one of the main aspects that is not specified in this solution is the interface between vehicles and CAs. Next, we present the changes to the European vehicular PKI that we assumed in order to define our vehicular PKI.

2.6 Summary

In this chapter we have studied the state-of-the-art PKIs used in the identity management of vehicles during V2X communications in Europe and America. We started by overviewing the architecture of such solutions, then approached some of the operational aspects needed to support a V2X environment. We Finalize this chapter by providing detailed overview of the standards behind the European PKI, and the existing ITS simulators. In the next chapter, we present our V2X environment which is based on the European PKI and standards. We start by overviewing its architecture, then we detail its implementation.

Chapter 3

Proposed Solution

In this chapter we start by providing an overview of the system's architecture. For each of the components, we analyse their basic functions and how they connect with each other. In the final part of this chapter we discuss the implementation of such components by describing the most important implementation decisions and the technologies adopted.

3.1 Overview of the System Architecture

Our solution is primary based on the European vehicular PKI and ETSI's standards. As such, the architecture of our PKI is similar to that of A Generic Public Key Infrastructure for Securing Car to-X Communication [12] that uses the certificates and message formats standardized by ETSI [6] to secure V2X communications, as specified in Section 2.1.1. In regards to the software needed to support such PKI, Figure 3.1 provides an overview of its main components.

3.1.1 System Components

V2X Library The V2X Library is designed as a software library package. Its main goal is to create all the data structures specified in the latest version of the ETSI TS 103 097 standard. Such structures combine to form the certificates and messages that are used by the vehicles and PKI. In addition to this, the library also allows is users (PKI Manager and Vehicle Manager) to perform cryptographic operations related with the generation of certificates, signature of messages as well as their verification. The usage of this library allows the PKI Manager and Vehicle Manager to perform such operations in conformance with the approved standards and algorithms.

PKI Manager The PKI Manager is a web application that functions as a backoffice to the PKI. Such application aims to provide administrative services related to the creation and storage of the vehicular PKI. For example, it enables an admin user to log in and create new CAs, their cryptographic keys and certificates. The PKI Manager allows the creation and configuration of a new PKI or even change the
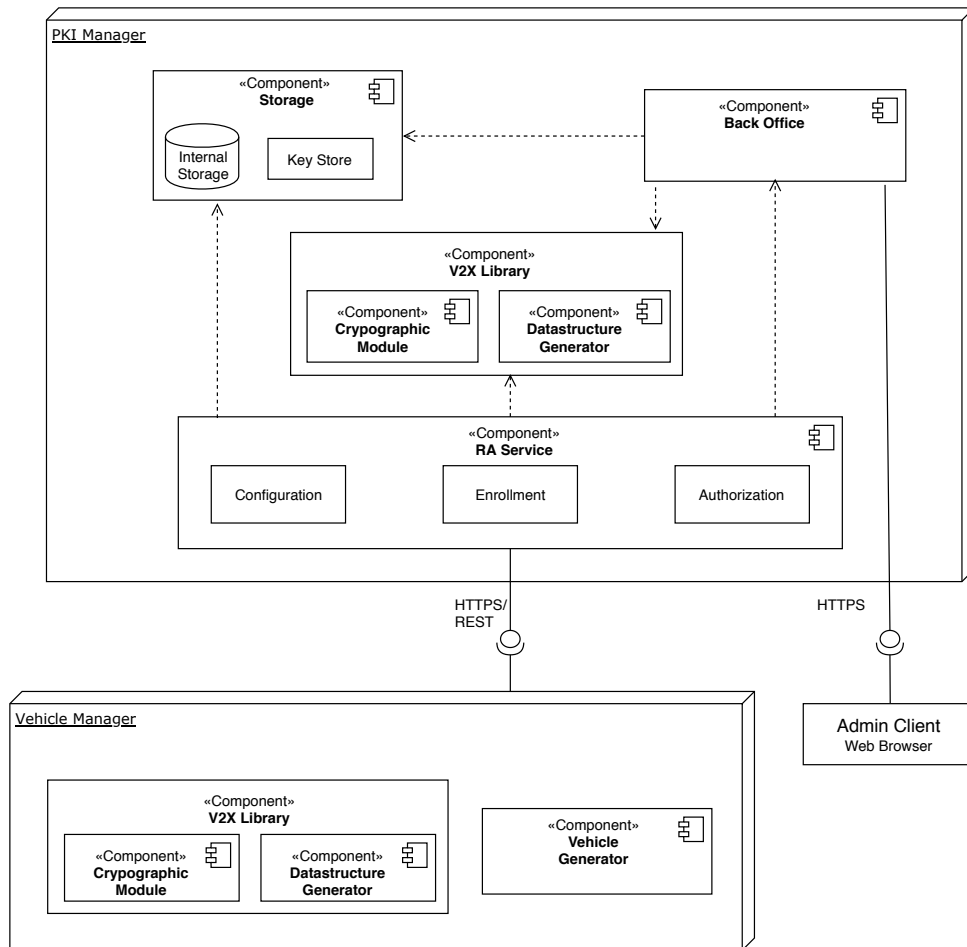
Figure 3.1: Overall system architecture.

structure of an already existent PKI. This application is connected to a database and KeyStore to provide persistence regarding the PKI information.

RA Service   The RA Service is designed as an API (Application Programming Interface) of the PKI Manager, its main goal is to act as proxy between the vehicles of the Vehicle Manager and the CAs existing on the PKI Manager. The RA has two responsibilities: verifying the vehicle's identity and supporting their requests for enrollment certificates and authorization tickets. The former task requires the RA to perform an initial vehicle configuration, much like authentication, the goal is to "remember" and securely identify each connection to an already configured vehicle. The later task involves sending such requests to the correct CAs for certificate issuing and responding to the vehicles with the requested certificates. The RA Service uses encryption and digital signatures to ensure that the exchange of certificates is secure and the privacy of the end-entities is protected.

Vehicle Manager   As the name suggests, the Vehicle Manager aims to manage the vehicles that will participate in V2X communication. This application runs on its own process and can be configured to create a given number of vehicles each as a client of the RA Service. Once created, the vehicles can contact the RA Service in order to request the end-entity certificates. Only with this initial configuration

done, the vehicles are able to start communication with each other in respect to the Vehicle Manager configuration.

3.1.2 Communication

Now that we have seen components of the project and what they do individually, it is time we study how they interact with each other and how is the communication organized. As we can see from Figure 3.1 we assume a client-server architecture, where the server is the PKI Manager and the Vehicle Manager acts as the client. On the server-side, the client requests will enter through the RA Service API, which then communicates with the back end and V2X Library to access the PKI services. As the RA Service is part of the PKI Manager project, the communication with the back end is achieved through simple service calls. Regarding the V2X Library, because this component exists on a different project we used it as a dependency of the server. The V2X Library is a Maven project, so it is possible to include it as one of the server's dependencies, which gives the server access to all the public classes and methods that the V2X Library has to offer. With this information is possible to create an interface on the server-side specifying the methods that will be used by the server as services of the Library. For example, generate a certificate or message, or verify them. Regarding the communication between the server and client, the HTTPS protocol is used to ensure protection of the sensible data that will be exchanged between them. Within the client, the same process is used for the usage of the V2X Library's services.

3.1.3 Protocol

The interaction between the Vehicle Manager generated vehicles and the PKI is divided into several phases. For the definition of such phases, we assumed the vehicle's life cycle as defined in Section 2.1.2. Essentially, vehicles use the information installed at manufacture time to request the enrollment credential, later using this credential in order to request authorization tickets to start V2X communications. As seen in Figure 3.1, the only way that a vehicle can reach the CAs on the PKI Manager is through the RA Service. Specifically through the RA's services of **configuration**, **enrollment** or **authorization**, each of these services represents a phase in the vehicle to PKI interaction. The SSL protocol is used to secure the integrity, privacy and authenticity of the sensitive data that is transmitted during by the Vehicle Manager and RA Service communications.

Vehicle Configuration

The first phase is the configuration. This phase aims to support the vehicle and RA configuration at vehicle manufacture time. Specifically, when the Vehicle Manager is generating new vehicles it uses this service in order to register them and to provide such vehicles with the information regarding the PKI. As we can see from message 1 to 2 represented on Figure 3.2, first the Vehicle Manager starts this phase by sending messag1 containing the canonical public key and ITS identifier of a newly generated vehicle. The RA stores this information in its database and responds with the the PKI information. The secure communication channel between the Vehicle Manager and RA Service allows the both RA to
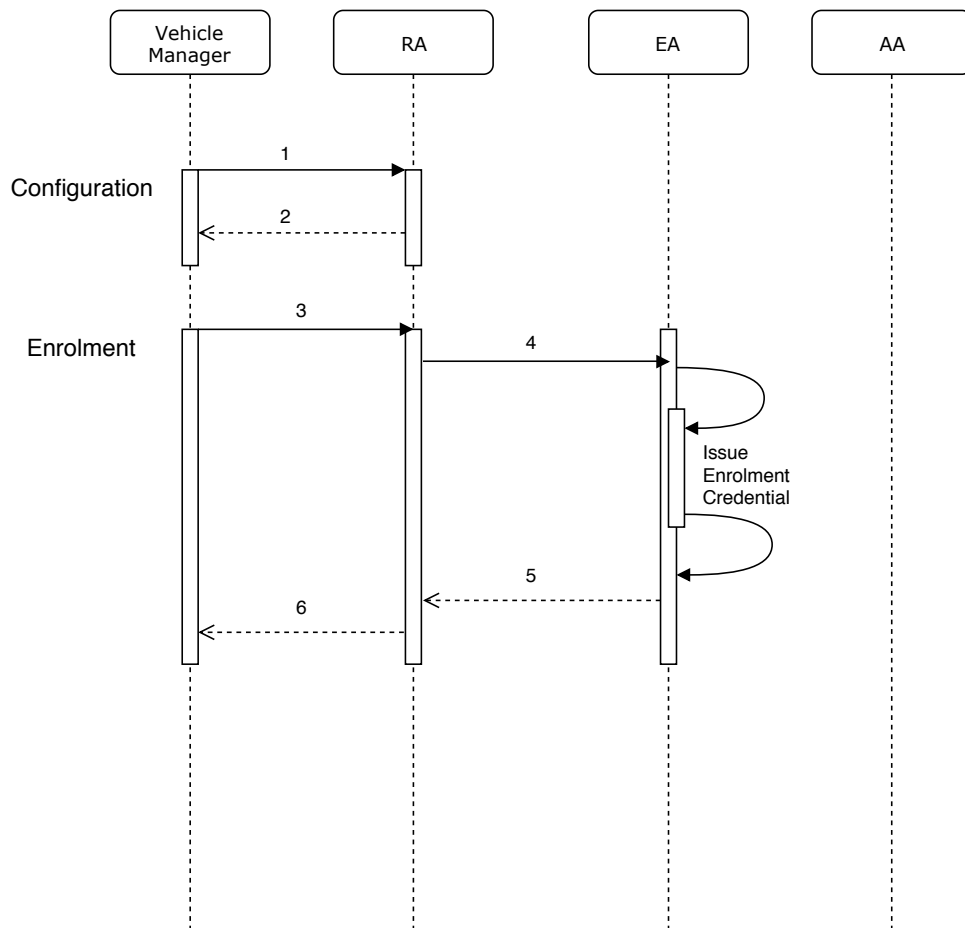
Figure 3.2: Vehicle configuration and enrollment process flow.

keep track of the trustworthy vehicles, and the vehicles to trust the PKI information which is responded by the RA. Such configuration will be the base of the trust that the RA has on the client vehicles for future interactions.

Vehicle Enrollment

The second phase is the enrollment. Vehicles which have completed the initial configuration can use this service to request the enrollment credential. To secure this type of communication we provide security at two levels: at the channel level through the SSL protocol and at the application level through the usage of digital certificates, signatures and encryption. Messages 3 to 6 represented on Figure 3.2 depict the communication steps of this service. The first step is made by the vehicle, in order to request the enrollment credential it first builds an enrollment request structure which is signed with the vehicle's private canonical key, and is encrypted for the attributed EA, see Section 3.2.2 for more information about the enrollment request structure. This request is the base of the application provided security. The second step involves the vehicle sending such request along with its secret identifier (ITS identifier) and the name of the attributed EA to the RA on message 3 through the SSL channel. Because the request itself is encrypted to the EA, the RA upon receiving it is not able to associate the vehicle's identifier with its future enrollment credential. Even so, the RA is able to perform the first vehicle identity

verification. This is done by validating if the vehicle is already configured and is stored on its database. If this is the case, then the RA will send message 4 to the target EA. This message contains the encrypted request and the vehicle's public canonical key which will allow the EA to perform the second vehicle authentication verification. Unlike the first validation done by the RA which depends mostly on the security at the channel level, the second validation depends essentially on the security provided by the application. To do so, the EA first decrypts the request and validates the canonical signature made by the vehicle. Only if the signature is valid, it will issue the enrollment credential and include it in the response message 5. Such response is signed by the EA, encrypted to the vehicle and then sent to the RA which has the responsibility of returning it to the original vehicle using message 6.

Vehicle Authorization

The last phase is the authorization which can be started by the enrolled vehicles. As the last phase, the security of this interaction is assured by the secure communication channel between the Vehicle Manager and PKI Manager, and by the application layer. Each time a vehicle requests one authorization ticket to an AA its enrollment must be verified. However, if such validation is performed by the AA the privacy of the vehicle would be compromised as the AA would have access to both the vehicle enrollment information (identity) and the pseudonym authorization ticket. To protect the vehicle's privacy, the enrollment validation is done by the EA which issued the vehicle's enrollment credential. Since a vehicle needs more than one authorization ticket, the consequence of this added privacy is the need to send more messages decreasing the performance of the vehicle authorization process as a whole.

As we can see from Figure 3.3, we have two flows from which a vehicle can request an authorization ticket. This enables us to leverage the RA in order to speed up the authorization process for each vehicle. The authorization process for a vehicle consists of multiple requests, if the vehicle is requesting the first authorization ticket the enrollment will be validated by the EA, otherwise the RA informs the AA that the vehicle's enrollment was previously validated. For the first authorization request, the vehicle starts by building the authorization request structure, such request contains all the information that the EA needs to validate the vehicle's enrollment status and that the AA needs in order to issue an authorization ticket. To ensure security and privacy in this type of communication, the vehicle's enrollment information such as its enrollment signature is encrypted to the EA and the authorization information encrypted to the AA. The vehicle sends message 1 containing the authorization request to the RA which knows how many requests the vehicle will perform during its authorization process. Message 2 shows the request being sent to the AA which decrypts it and sends the enrollment information to the EA in message 4. The EA decrypts its part of the request, validates the vehicle's enrollment signature, using the vehicle's enrollment credential (referenced by the request), and validates if such certificate is valid. If the verifications are in order, the EA sends message 4 which notifies the AA that the vehicle is authentic. The AA then issues the Authorization ticket, builds an authorization response structure which is signed, encrypted to the vehicle and sent to the RA within message 5 as a positive authorization.

At this point the RA knows that the vehicle is enrolled and that the authorization was successful, and is able to return the response to the vehicle in message 6. For the remaining requests the vehicle
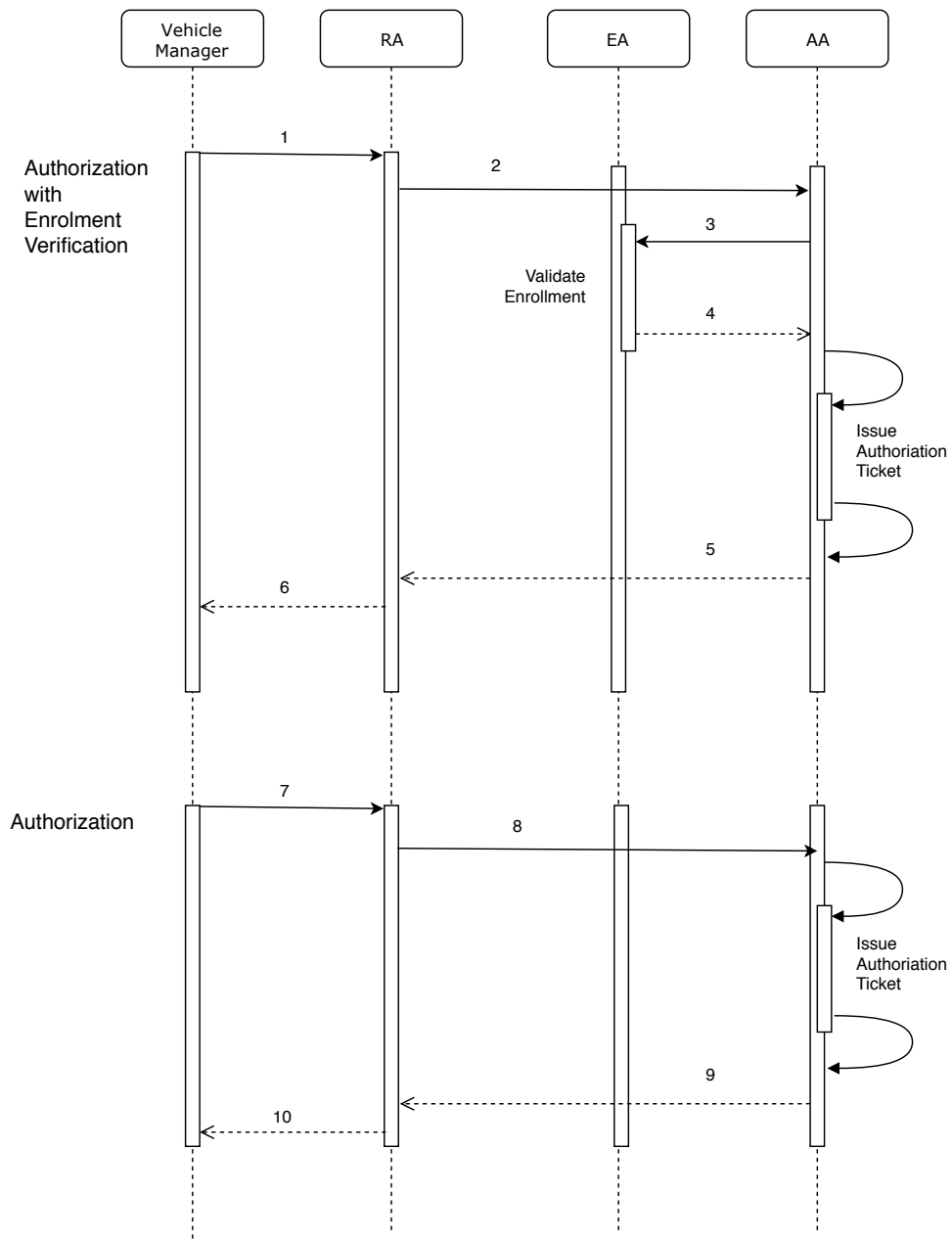
Figure 3.3: Vehicle authorization process flow.

builds the authorization request and sends it to the RA as usual, as seen in message 7. The RA knows that the vehicle's enrollment has been validated for this authorization process and sends the request to the AA in message 8, the AA decrypts the requests, validates the authorization information, issues the authorization ticket, and returns the response to the RA in message 9. In the last step, message 10 shows the RA returning such request to the vehicle which stores the requested certificate. The same flow is repeated until the vehicle has all the authorization tickets for the time specified by the RA.

3.2 V2X Library

As discussed before, one of the most basic concerns of this work is to implement a V2X ecosystem that is compatible with the most recent standardization efforts done by European organizations. In order to satisfy this requirement, the V2X library was implemented with ETSI's standards in mind. However, instead of blindly coding the information specified on the European standard we first searched for an already existing implementation. We found such a solution on the GitHub repository named C2C-Common [30], an open source Java package. However, upon closer inspection we noticed that C2C-common does not support the latest version of the European standard (1.3.2 at the time of this writing). In order to not reinvent the wheel, we decided to extend C2C-common by adding the necessary changes. In this section, we start by describing the detailed architecture of the library, then we take a look at the implementation of some of its most important data structures, and finalize by describing the services that our library provides.

3.2.1 Detailed Architecture

The architecture of the V2X Library can be described as a layered structure, where at the highest level we can find the classes which allow the generation of the more specific data structures such as the EA, AA and Root CA certificates; vehicular authorization and enrollment certificates; and the V2X messages such as CAMs. As we go down a level the can find the more general substructures which the higher level structures depend. Finally, at the lower level we can find the classes which enable the transmission of such data structures in a cross-platform way.

As we can see from Figure 3.4 the architecture of the V2X Library is divided in three layers. Starting at the layer 1 we have the super classes which are responsible to implement the Abstract Syntax Notation 1 (ASN.1) Canonical Octet Encoding Rules (COER) as used by the ETSI standard. The ASN.1 is a standard description language for defining data structures that can be serialized and deserialized in a cross-platform way. ASN.1 is closely coupled with a set of encoding rues (e.g. COER) which specify how to represent a data structure (e.g. an integer) in a series of bytes (serialization) and vice-versa (deserialization). At this level, we can find the most basic of COEREncodable structures such as sequence, enumeration, choice, etc. as well as their implementation of the encoding/decoding methods. Such structures will be part of the V2X certificates and messages according to the specification of ETSI. Because the definition of these structures and the respective encoding/decoding implementations

25

depend only on the version of ITU-T X.696 standard [31], change on the C2C-common library at such a low level was not needed. In addition to the implementation of the COER structures, it is at this layer that the V2X has performs the most basic cryptographic operations. Such operations include the digest, signature and encryption of data and will be used exclusively by the layer 2 in order to build the data structures which require them. For example, certificates and V2X messages are always signed, and certificate requests are both signed and encrypted.

Going up a level we can find the sub-classes of the more general COER structures, for example, the certificate which is a child of COERsequence. This means that a certificate is just a sequence of fields (i.e. base data structures such as the host name, issuer identifier, duration validity, signature, etc) and should be encoded/decoded as such. Because the certificate and message structures depend on the version of the ETSI standard, it was at this level that we introduced more changes to this library.

Besides updating C2C-common, our contribution also included the addition of a new set of standardized messages, the request for vehicle certificates. Such messages are encrypted and signed to assure confidentiality and authentication on the certificate request and deliver. In our case, they are used by the vehicle manager and RA. The specification of this data structures can be found in the ETSI's Trust and Privacy Management standard [15] and follow the same COER encoding rules.

At the layer 3 we can find the generator classes that the library user needs to instantiate in order to generate certificates, V2X messages and certificate requests. The generator classes are responsible to implement the certificate profiles for the different subjects and the different types of V2X messages for the vehicles. In addition to the generators, this layer also provides to the user the higher level cryptographic operations such as the validation of a V2X message, certificate, generation of a new keypair, etc. In our case, the generator classes and the user cryptographic module will be used by interfaces present on the Vehicle Manager and PKI Manager.

To assure that every data structure is correctly implemented unit tests were performed to test the serialization and deserialization of the objects.

3.2.2 The Data Structures

We have seen how the library is organized, now we take a look at the implementation of the most important V2X structures. We start by describing the lower level COER structures, then we describe the higher level structurs such as the certificate requests.

COER Structures

Before we can understand the higher level data structures, we have to understand the lower level COER structures which are part of them and therefore are the base of this library. All implemented data structures derive from the same type, the *COEREncodable*, which allows us say that each can be encoded or decoded according to the *COER* rules. In terms of implementation, this is enforced by the *COEREncodable* interface which declares two methods, one for encoding and the other for decoding. This interface provides us with a solid base, as any class that implements it is able to provide its own implementation
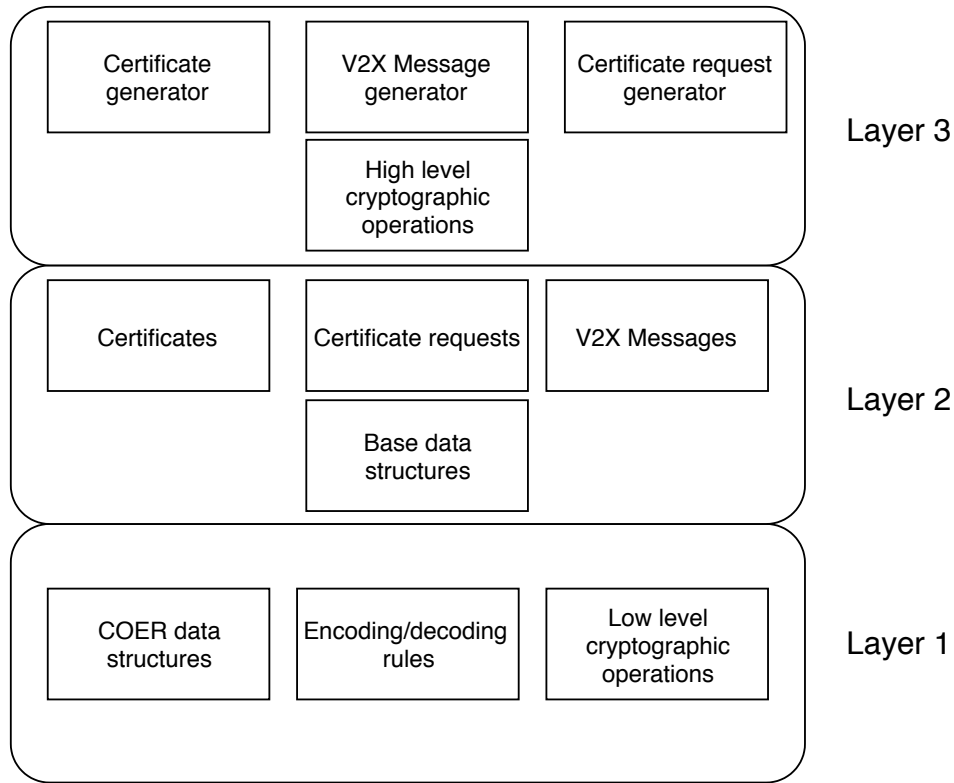
Figure 3.4: Layered structure of the V2X Library.

of the two methods. The classes which implement this interface are one level higher than the base type, they are the *COER* structures. In this Section we describe the implementation of the most complex of such structures, the *COERSEquence* and *COERSChoice*. To understand the relation between the classes, Figure 3.5 provides an UML classdiagram.

COERSequence The *COERSequence* is a class which implements the encoding and decoding of a sequence of *COEREncodable* elements. This class is very useful in the V2X Library as it allows us to combine a set of elements in a specific order to form a higher level data structure like a *EtsiTs103097Certificate*. Generally speaking, the *COERSequence* is a super class which provides the implementation of the *encode* and *decode* methods. Such class can be extended in child classes which can be seen as more specific types of *COERSequence*, for example, the *EtsiTs103097Certificate* which is an ordered sequence of certificate fields, and *HeaderInfo* representing a specific sequence of header fields on a V2X message. Once an object of the sub class is instantiated it is possible to call the *encode* or *decode* methods to assure that the encoding rules are respected according to their definition on the super class.

In terms of implementation, the *COERSequence* class represents an ordered sequence by having a list of *Components* as a class attribute. A *Component* is a helper class that represents a single component within the sequence. This class contains the to be encoded *COEREncodable* value; a boolean variable representing if this value is optional; and an empty *COEREncodable* object which indicates the value's type in the decoding logic. This list of components represents the *COERSequence*

27
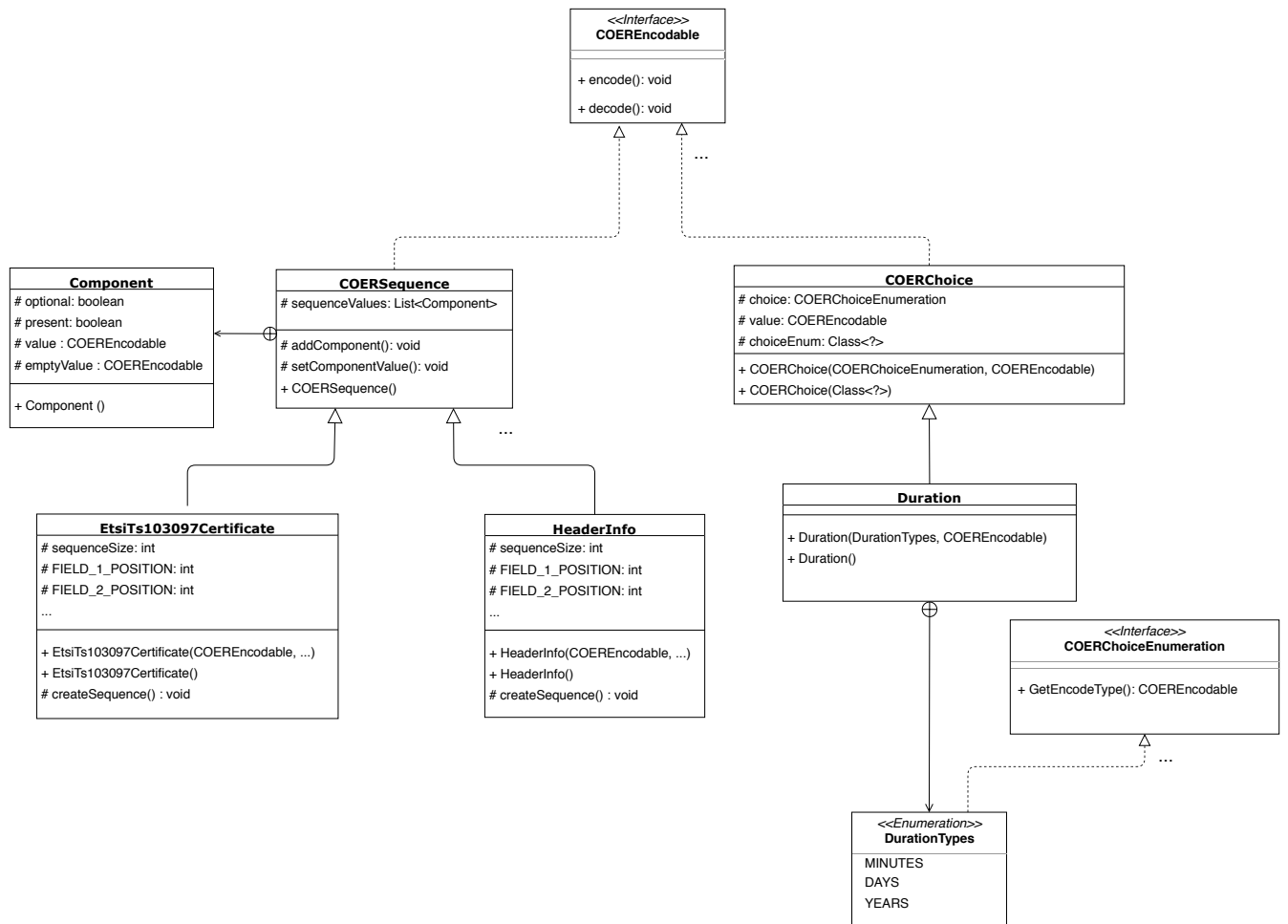
Figure 3.5: Class diagram of the V2X COER structures.

and is initialized by the class's constructor which receives the sequence's size in the parameters. The other classes that extend the *COERSequence* are able to interact with such list during the process encoding or decoding an object.

Before a sequence can be encoded it must be first populated, the *COERSequence* class provides two methods for this effect: the *addComponent* and *setComponentValue*. The first method has the responsibility of creating the frame of the sequence, which is relevant to the decoding process to indicate the specific component type, position in the sequence, and whether the component is optional. The second method can be posteriorly called to add the sequence values to the frame, completing the creation of the sequence. It is possible to add a *null* value in the sequence when the value is optional and not present. To minimize usage errors, *setComponentValue* always checks if the mandatory components are present.

COERSequence Encoding   The encode process starts when creating a sub class of *COERSequence* (e.g *EtsiTs103097Certificate*) using its dedicated encoding constructor with the sequence values as parameters. Each sub class maintains its own frame information such as the sequence size, component order and optionality. Because of this, when an object of the sub class is created for encoding the constructor is able to initialize the list of *Components* within the super class. This process is achieved

in three steps: first the super constructor is called with the sequence size, then the *addComponent* and *setComponentValue* methods are called for each sequence element with the respective frame information and value. Once the sub class object is created, it is possible to encoded it by calling the *encode* method. This method receives a *DataOutputStream* in which the encoded bytes will be written to. The encoding of a *COERSequence* consists of a preamble followed by the encoding of each sequence element. The preamble is a stream of bits which allows specifying the presence of the optional elements in the sequence. Each present element (not null) is then encoded with respect to its type (e.g COERInteger, COERString, Signature, etc) in the order that they appear in the sequence.

COERSequence Decoding   The decoding process is similar to the encoding. Instead of calling the encoding constructor we call the decoding constructor to create the frame of the sequence within the super class *COERSequence*. This is achieved by the decoding constructor in two steps: calling the super constructor with the sequence size, and the *addComponent* method for each sequence element with its type, order and optionality. The result is the list of *Components* on the super class containing the frame of the sequence but no values. We now have a *COERSequence* sub class which is instantiated to be part of the decoding process. To do so, we just need to call the *encode* method with the *DataInputStream* in which the encoded values will be read from. The process of decoding starts by reading the preamble to identify which of the optional values are present. The next step is to decode each present sequence value into their original object. This is done by iterating through each member of the empty list of *Components* and decoding it according to its type (represented by the empty *COEREncodable* value). The final step is to set the *Component* value to the obtained object. When the decode process finishes, the sub class will have access to the complete sequence in the *COERSequence's* class list attribute.

COERChoice   The *COERChoice* is a class that implements the encoding and decoding of a choice of *COEREncodable* elements. This class is very useful in the V2X Library as it allows us to specify the choice of one element from a set of known *COEREncodable* values. Some applications include: the type of time duration on a certificate's validity (years, days etc.), the certificate's issuer identifier (self signed or digest of the issuer's certificate), and many others.

Like the *COERSequence* and all other COER classes, the *COERChoice* is a super class which provides the implementation of the *encode* and *decode* methods. The process of encoding or decoding a choice starts when creating a child of the *COERChoice* (e.g. *Duration*) using the dedicated constructors.

The objective of the encoding constructor is to initialize all the attributes on the super class (*COERChoice*) needed for encoding. Such attributes include the choice and its value. To implement this, all sub classes contain a Java enumeration listing the possible choices for its type, for example *Duration* contains an enumeration listing the possible types of time duration (microseconds, seconds, days and years). With this enumeration it is possible to instantiate a sub class object by calling the encoding constructor and passing an item which represents the choice and its corresponding *COEREncodable* value.

To uniformize the type of these enumerations through all the *COERChoice's* sub classes, they implement the same *COERChoiceEnumeration* interface. This enables the encoding constructor of the sub class to call the corresponding constructor on the *COERChoice* passing the *COERChoiceEnumeration* item and the value. An example of this interaction would be creating an object of *Duration* with the *COERChoiceEnumeration* item of *YEARS* and the value of 4, representing a duration of four years. In this case the *Duration* class encapsulates the value in to a *Uint16* structure, an unsigned 16 bit *COEREncodable* integer, and calls the *COERChoice* encoding contrustor with the choice of duration (time unit) and the value. Thus, initializing the super class attribues for encoding and concluding the creation of the *Duration* object.

COERChoice Encoding   To encode the created sub class object we simply need to call the *encode* method, the implementation of this method exists in the super class because it is the same for all types which extend the base type of *COERChoice*. In terms of implementation, the *encode* method works with the attributes of the choice (*COERChoiceEnumeration*) and value (*COEREncodable*). The encoding consists in the creation of a tag followed by the encoding of the *COEREncodable* value. *COERTag* allow the decoder to know the specific choice that the value belongs to, as it contains the position of the choice within its *COERChoiceEnumeration*. This stream of bytes is enabled by the *getOrdinal* method which is implemented within all *COERChoiceEnumerations*, thus allowing the *COERChoice* class to call it using its choice attribute. The *encode* method then creates and encodes a *COERTag* object with the ordinal value, encodes the *COEREncodable* value (in the class value attribute), and finally appends it to the tag. The result is a stream of bytes representing the choice and the value.

COERChoice Decoding   The process of decoding a sub class of *COERChoice* is similar to the encoding, as it starts by creating an object of the sub class but calling its decoding constructor. Such method will initialize the choice attribute on the super class so that the value can be filled by the decoding process. This is achieved by calling the corresponding constructor on the *COERChoice* class with the main type of the *COERChoiceEnumeration*. For example, in the case of the *Duration* class is the *DurationTypes* enumeration which implements the *COERChoiceEnumeration* interface. This will initialize the *choiceEnum* attribute on the super class which is needed for the decoding process. Once the empty object is created (i.e. without the value), the *decode* method can be called. Such method will first decode the *COERTag* to get the specific choice from its *choiceEnum* attribute. With the choice it is possible to get the *COEREncodable* value associated to it, we just need to know its main type. This is achieved by the *getEncodableType* method which is implemented within the *COERChoiceEnumerations*, thus enabling the *decode* method to use it on the recently discovered choice attribute. This method contains a switch that given the choice will return its intended main type. For example, in the *DurationTypes* enumeration it will return an empty object of *Uint16* since this is the type that represents the number of years, minutes, seconds, etc in a time duration. At this point the *decode* method knows all the pieces to the puzzle and is able to decode the value by calling the *decode* method for its main type. The application of this method will update the choice and value attributes of the *COERChoice*,

thus concluding the decoding.

Certificate Requests

Besides updating the implementation of the certificates, V2X messages and all other data structures which they depend on, we had to implement the requests for vehicle certificates and their corresponding response. The specification for these data structures can be found on the ETSI's Trust and Privacy Management standard [15] and they depend on the same *COEREncodable* data structures as the V2X messages.

Enrollment Request   The first message that we implemented was the *EnrollmentRequest* shown in Figure 3.6. This request contains two signatures and is encrypted to ensure the confidentiality, integrity and non-repudiation of the vehicle's important information such as the vehicle id, requested certificate attributes and the public verification which will be certified.

The request for an enrollment certificate is sent by the vehicles to an EA. In order to protect the privacy of the vehicles by hiding the vehicle's enrollment data such as ITS identifier this message will be encrypted with a symmetric key that is to be shared with the recipient EA. In terms of implementation, this message is an *EncryptedData* structure which is a sequence that contains two elements: the recipient information and the ciphertext. The recipient information is the element that allows us to share the encryption symmetric key with the EA. This structure is implemented as a *PKRecipientInfo* (Public key recipient information) containing the hash of the certificate that belongs to the recipient EA (*recipientId*), and the 128 bit AES key encrypted with ECIES using the encryption public key of the recipient EA. The symmetric key is shared with the EA for of performance reasons, since encrypting large amounts of data with symmetric cryptography is faster than using symmetric cryptography. For interoperability reasons, the encryption algorithms used are the ones approved by the standard. The ciphertext is the request's outer signature structure.

The outer signature exists to prove that the requesting vehicle has possession of the canonical keypair which was attributed to it when it was manufactured or the keypair of a current valid enrollment certificate. This allows the EA to trust that the origin of this request is an authentic vehicle, assuming that such key pair was securely stored by the vehicle's OBU. In terms of implementation, the signature is contained in a *SignedData* structure which is sequence that also contains fields relevant to the signature's validation as well as the signed data. Specifically, this structure contains: the id of the hash algorithm used, signed data, signer information, and the signature itself. The signature over the to be signed can be calculated by two district ways depending on the vehicle's enrollment state: if this is the first enrollment request, the signature should be done with the vehicle's private canonical key; else it should be done with the private key of the currently valid enrollment certificate. To indicate the signature method, the field signer identifier is a *COERChoice* which contains the hash of the enrollment credential in the case re-enrollment or empty otherwise. The data protected by the signature (signed data) is a sequence that contains a header and payload. The header is the data structure which contains information needed for the validation of the outer signature, it contains the *Provider Service Identifier* PSID and
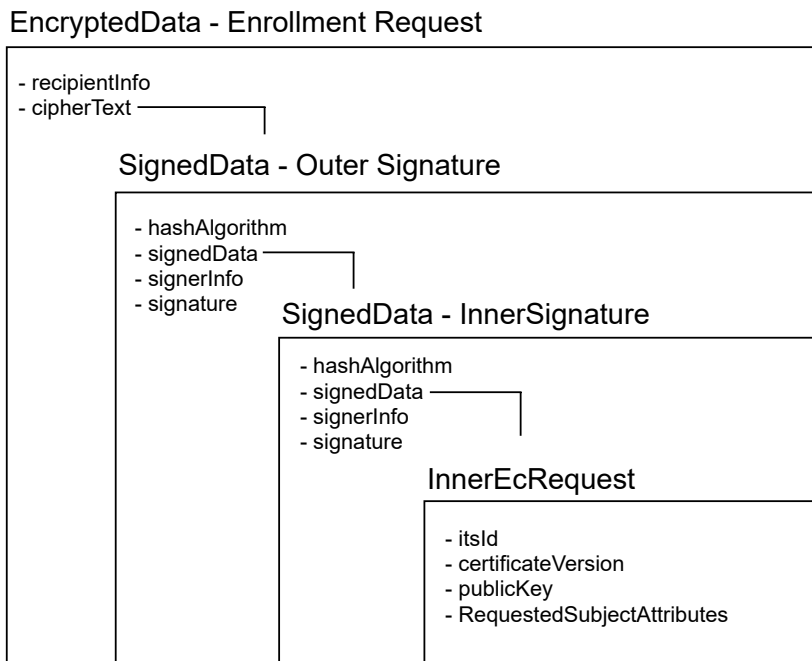
Figure 3.6: Enrollment request message.

the signature generation time. The PSID is a *COERInteger* which identifies this message as a secured certificate request as assigned in the ETSI TS 102 965 standard [32]. The payload of the outer signature contains the request's inner signature.

While the outer signature serves to prove the identity of the vehicle, the inner signature serves to prove that the requesting vehicles has position of the verification key pair which will be part of its enrollment certificate. I terms of implementation, the inner signature is very similar to the outer signature as it depends on the same sub structures. The only difference is that this signature is done using the vehicle's private verification key corresponding to the public key to be sent by this request. Therefore, the signer information field is always empty, indication that this is a self signed signature. The payload contains the data relevant to the emission of the enrollment certificate, this data is wrapped in a *InnerECRequest* structure which is as a sequence containing: the vehicle's identifier, the version of the certificate standard, the public verification key, and the requested subject attributes. The requested attributes are the desired enrollment certificate attributes and contain data such as the enrollment validity period, region, desired application permissions, and the certificate subject name.

Enrollment Response The second request that we implemented was the *EnrollmentResponse* shown in Figure 3.7. This message will be sent by the EA to the vehicle in response to its enrollment request. To protect the vehicle's privacy and securely deliver the enrollmnet certificate this request is signed and then encrypted. The request is encrypted using the symmetric key which was shared by the vehicle through the enrollment request. To indicate this, the recipient information is implemented as a *PreSharedKeyRecipientInfo* structure which contains the hash of the AES key. The ciphertext of the encrypted data contains the enrollment response signature, which is done using the private key that is paired with the EA certificate's verification public key. This signature allows the vehicle to validate
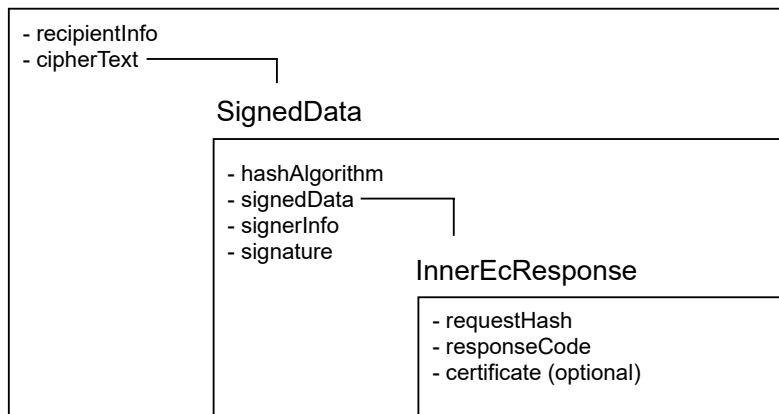
EncryptedData - Enrollment Response

Figure 3.7: Enrollment response message.

that the response originated from the correct EA. The signature protects the *InnerECResponse* which contains the original request's hash value, a response code, and possibly the requested enrollment certificate. The hash value allows the vehicle to map this response to the original request by calculating and comparing its hash value. The response code indicates whether the request was successful or not.

There are many possible ways which can cause the EA to decline the enrollment request, for example, if it can't parse the request, can't' decrypt, the EA is not the recipient, invalid vehicle signatures, denied permissions, etc. To inform the vehicle of the problem, *EnrollmentResonseCode* is a COEREnumeration which encodes the possible outcomes of the EA's enrollment validation. Only in the positive case the *InnerECResponse* provides the enrollment certificate.

Authorization Request   After we implemented the enrollment request and response we implemented the *AuthorizationRequest* shown in Figure 3.8 since it depends on the vehicle's enrollment certificate. The authorization request is sent by enrolled vehicles to an AA which sends the enrollment certificate to the EA for vehicle enrollment validation. Since the authorization request includes the vehicle's enrollment certificate it introduces privacy concerns regarding the ability of the AA to link the enrollment certificate to the issued authorization tickets. To protect the vehicle's privacy the authorization request can be divided in two parts: the part which is encrypted to the AA and the part which is encrypted to the EA.

The process of encrypting the request for the AA is similar to the one used in the enrollment request since we are encrypting data to a certificate recipient. The authorization request is an *EncryptedData* structure containing the recipient information and the ciphertext. While the recipient information contains the shared symmetric key which decrypts the request. The encrypted data is a *SignedData* structure which ensures the AA that the vehicle is in possession of the verification key pair which will be certified by the authorization ticket. The signed data within this structure is the *InnerATRequest* which contains the information that the AA needs to issue the authorization ticket and the information that the EA needs to validate the vehicle's enrollment. The *InnerATRequest* is a sequence that contains: the verification public key, a *ecSignature*, and *sharedATRequest* structures. The *sharedATRequest* is a structure which is to be shared between the AA and EA, it contains information that needs to be validated by both authorities in
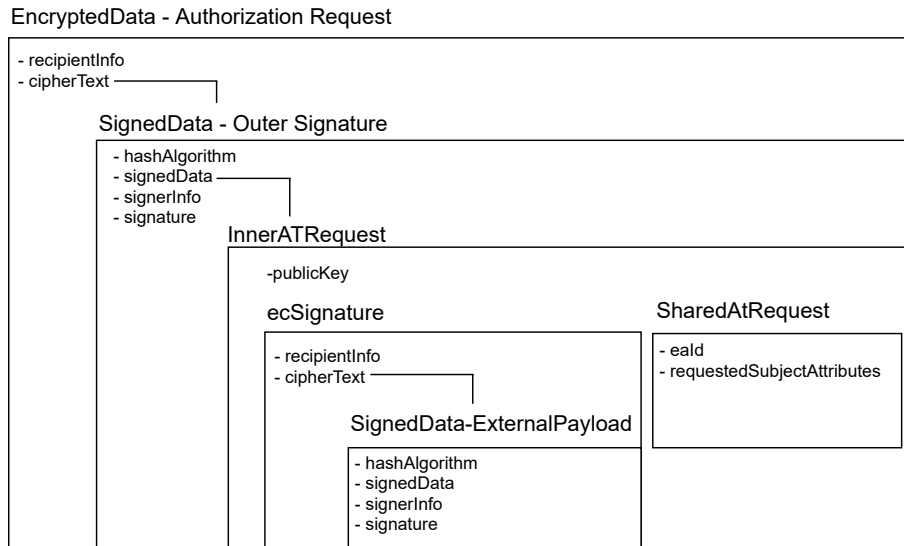
33

Figure 3.8: Authorization request message.

order to issue the authorizatio ticket. This structure contains the *eaId*, allowing the AA to know which was the EA that enrolled the requesting vehicle; and the requested subject attributes which will be validated by the authorities. The *ecSignature* contais the vehicle's enrollment credential signature, which is the part of the authorization request encrypted to the EA to ensure privacy in the enrollment validation. The ciphertext contains the signature computed with the private key corresponding to the enrollment credential of the vehicle. This mechanism allows the EA alone to decipher its part of the request and validate the vehicle's enrollment. To specify to the EA which enrollment credential was responsible for the signature, the field signder identifier contains the hashed vehicle's enrollment certificate. The signed data contains the hash of the previous *sharedATRequest* since this structure is to be seen by both the EA and AA, therefore it can not be directly included in the encrypted *ecSignature* structure. This external payload mechanism on *ecSignature* allows it to ensure the integrity and non-repudiation of the *sharedATRequest* while hiding vehicle's enrollment credential to the AA and still allowing it read the *sharedATRequest*.

Authorization Validation Request and Response   These request/response messages are the intermediate step in the vehicle's request for an AT. To validate the vehicle's enrollment an *Authorization-ValidationRequest* is constructed by the AA and sent to the EA. This structure contains elements from the Authorization request which was sent by the vehicle to the AA. Specifically, the *ecSignature* and *sharedATRequest*. The response is sent by the EA to the AA and contains the response code which will determine whether the AA is allowed to issue the AT or not.

In terms of implementation the response is a *AuthorizationValidationResponse* structure which is a sequence that contains the request hash, response code and the confirmed subject attributes. The request hash is SHA-256 digest of the *AuthorizationValidationRequest* which will allow the AA to associate this response to its original validation request. The response code indicating the result of the EA's internal vehicle enrollment validation. In the positive case, the field confirmed subject attributes contains the
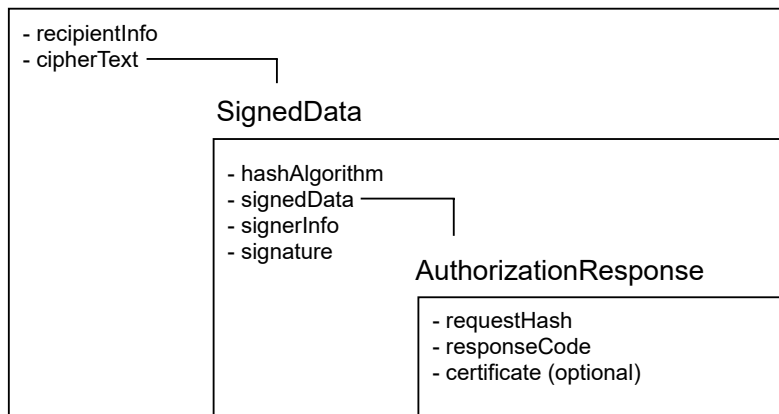
EncryptedData - AuthorizationResponse

Figure 3.9: Authorization response message.

subject attributes that the EA wishes to confirm; in the case that the EA denies the vehicle's enrollment the response code contains the reason, and the confirmed subject attributes is empty.

Authorization Response   The *AuthorizationResponse* shown in Figure 3.9 is the last message to be sent in the vehicle's authorization request. This message is sent by the AA to the vehicle and may contain the requested AT. To protect this response the message is signed by the AA and then encrypted such that only the original vehicle is able to decrypt. The encryption is similar to the method used in the enrollment response since a symmetric encryption key was previously shared by the vehicle with the AA in the Authorization request. The cipher text is a *SignedData* structure which contains a signature done with the AA's private key that corresponds to its authority certificate. The signed data is the *authorizationResponse* which contains: the request hash, allowing the vehicle to associate the AT request; the response code, which indicates the result of the authorization process; and the optional AT that is only present in the successful authorization case.

3.2.3   The Cryptographic Tools

When privacy and security are requirements of a system, the use of cryptographic tools is essential. Such tools allow us to keep the information confidential thorough communication, to verify its integrity, freshness and authenticity.

The process of providing the secured data structures to the other system components implies that it is this library's responsibility to perform the needed cryptographic operations in order to generate them. To facilitate interoperability between other potential systems, it is essential that such operations are performed according to the algorithms described in the latest version of the IEEE 1609.2 standard, which are also used by ETSI 103 097.

At a low level, the V2X Library uses Bouncy Castle as a security provider for all of its cryptogtaphic operations. Such API pvoides us with the implementation of the cryptogaphic algorithms allwoing us to abstact from their complex implementation details when performing cryptographic operations. The V2X

library contains within its functionalities the following cryptographic operations:

- The generation of symmetric and asymmetric keys. This operation must be called providing one of the supported key generation algorithms.

- The signing of data. This operation must be called providing a supported elliptic curve signing algorithm, the data to be signed, the signing certificate, and the signing private key. The result is a *Signature* structure as defined in the European standard.

- Verification of a digital signature. This operation must be called providing the data to be verified, the *Signature* data structure, and the signing certificate with the correspondent certification chain. This function returns a Boolean value corresponding to the validation result.

- Asymmetric encryption of symmetric key. Encrypting large amounts of data with asymmetric keys is inefficient, this should be done using symmetric keys. However, in order to decrypt a symmetric encrypted message, its recipient must have access to the same secret key (i.e. symmetric key) which encrypted the message. This operation exists to assure confidentiality in the process of sharing a secret key with a specific recipient. It should be called providing the secret key that will be encrypted, a supported elliptic curve encryption algorithm, and the public key of the recipient.

- Asymmetric decryption of symmetric key. This operation must be called providing the encrypted symmetric key, the recipients private key and the elliptic curve decryption algorithm.

- Symmetric Encryption of data. This operation should be used to encrypt data with the *AES CCM mode* algorithm, to call it we need to specify the data to be encrypted, the secret key and a nonce. The last parameter is to be used as input to the initialization vector for the encryption algorithm.

- Symmetric Decryption of data. This operation must be called providing the encrypted data, and the decryption key and nonce which was originally used to encrypt the data.

All of these operations are available at the Crypto Manager which was implemented as the default cryptographic module of V2X Library. It is also possible to extend the Crypto Manager in the future by adding new operations or support for new algorithms.

3.3 PKI Manager

Certificate issuing is an essential part of this work, however due to time constraints we were not able to extend mPKI. Our solution to this problem was to create the PKI Manager, a web application that functions as a demo PKI.

The PKI Manager contains a backoffice which was implemented with the sole purpose of demonstrating the creation of vehicular certificates. However, it is much more limited comparing to mPKI. In the PKI Manager all the cryptographic operations related to the generation of keys, certificates and messages are done using software (V2X Library).

Such operations share the processing resources with the machine which the PKI Manager is deployed, which can cause the entire machine to slow down and decrease the performance of the PKI Manager. In terms of security, these operations are only secure as the rest of the machine. A single security flaw in the operating system can compromise the security provided by the software encryption. In addition to the constraints in performance and security, the PKI Manager is also limited in terms of functionality. Although it provides the most basic vehicular PKI operations such as the enrollment, re-enrollment and authorization of vehicles it does not support the revocation of the certificates which belong to the end entities and CAs though a CRL mechanism. In this section, we describe the application, the technologies adopted, the features that it provides, and our implementation decisions.

3.3.1   The Technology

The PKI Manager is a web application implemented using the Spring Boot framework. This framework produces a stand-alone application and aims to reduce the time spent configuring it. This is achieved automatically by Spring based on the dependencies added to the project. For example, if we add a dependency that relates to a database, Spring will attempt to auto configure the application for database access.

These applications run on an embedded container, which simplifies the deployment process of the web application. We simply need to press the "start button" and access the application at the address indicated by *http://localhost:8081/*. This is done by Spring, because we added the spring-boot-starter-web dependency it pulls the spring-boot-starter-tomcat automatically which in turn starts Apache's Tomcat Web Server when we run the main method.

Our web application is organized as a Spring Web MVC application, this framework is a module of Spring that specializes in aiding the development of web applications. It provides all the functionalities needed to receive HTTP requests, delegate their processing to other components, and finally, build a response. MVC is an acronym for model, view and controller each name represents a fundamental part of a Spring Web MVC application. In order to understand each part, I will describe the basic flow of an HTTP request from the point of view of the Spring MVC.

When we access an URL in the browser that sends an HTTP request to our server, for example our *http://localhost:8081/*, the framework will try to find a class that is responsible to deal with such request, delivering to it the data that originated from the bowser. This class is an application controller. The controller then passes the data to a model, which executes the necessary business logic such as validation, calculations or database access. The result of the operations is returned to the controller, which in turn returns the name of the view along with the data needed to render the web page. Finally, the framework tries to find the specific view that is responsible to process such data into an HTML page and returns it to the browser.

In regard to our application's identity management and user authentication we decided to use Spring Security since it is also a module of Spring. By using a configuration file, we can limit the web pages which the users and administrators can access while authenticated. This feature allows us to protect

services and information from unauthenticated parties. The PKI Manager authentication is done using the usual username and password method. In this approach, the server stores the username and the hashed value of the password in its database. In addition to storing user credentials, the database also stores the role of each user allowing us to differentiate the privileges for the different types of users (regular user or admin).

3.3.2 The Database

One of the concerns that we had when creating the PKI Manager was the persistence of the PKI information. We decided that even though this PKI exists for demonstration purposes, it would be important to store it in a non-volatile way. This decision allows us to keep consistency of the PKI even if the server crashes/restarts. In this section we describe the data access layer of the PKI manager, we start by introducing the technologies adopted, then we take a look at the design of the database.

The database technology The database service is provided by PostgreSQL, a relational database management system. A database server runs locally, to which our application connects at the start of its execution.

The database access is provided by another of Spring's modules, the Spring Data JPA. This framework aims to simplify the data access layer of the applications by reducing the code needed for its implementation. To do so it relies on the entities and repositories. The entities are classes which correspond to tables in a database and are created with the help of the Spring annotations. It is through these annotations that we can specify the name of the table, its columns, primary keys, the relations between tables, etc. While the JPA entities allow us to shape the diagram of the database, the repositories provide us with a way to interact with it using queries. To be precise, these repositories are divided in two parts, the Repository and the Repository interface. The Repository is a class implemented by Spring, it has direct access to the database and queries it. The Repository interface is an abstraction of the Repository in which the developer can add methods in order to specify the supported queries. This can be done in different ways, for example, it is possible to create a query from the method's name or using Spring annotation. In our case, we used the first method for the more simple queries, and provided Spring annotation containing a native SQL query to support the most complex of the database queries.

The database design To provide persistence with regard to the PKI Manager, we designed a database which keeps the information about the PKI entities and the operations of the PKI Manager. To minimize redundant data and facilitate finding information, we introduced relations between these PKI entities naturally. figure 3.10 shows the entities that we took into account, their characteristics and relations.

As we can see from the Figure, the database stores in its tables information to support the user authentication and management of the vehicular PKI. Regarding the PKI information, it has a table for the CAs, keys, certificates and regions.

The *ca* table contains generic CA information such the CA type (Root, AA, EA), group (Root CA or Sub CA), name, and country. It is possible for a CA to have one or more keys associated, this is
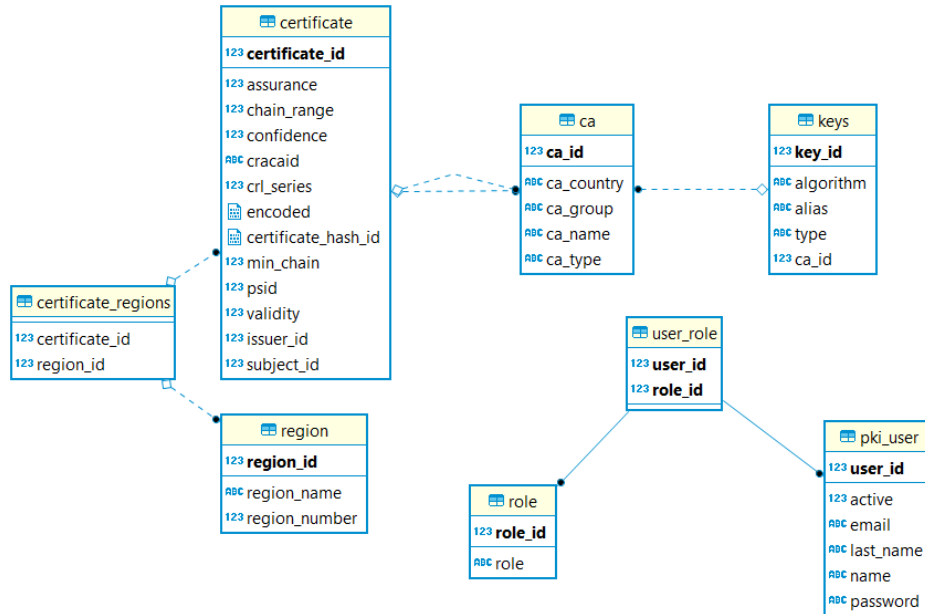
Figure 3.10: Database entity relation diagram.

expressed by the one-to-many relation between the *ca* and keys tables. Regarding the relations between CAs and certificates, we assumed two occurrences: A CA can be subject of a single certificate and the issuer of one or more certificates. These relations are expressed by the one-to-one and one-to-many relation between the *ca* and certificate tables.
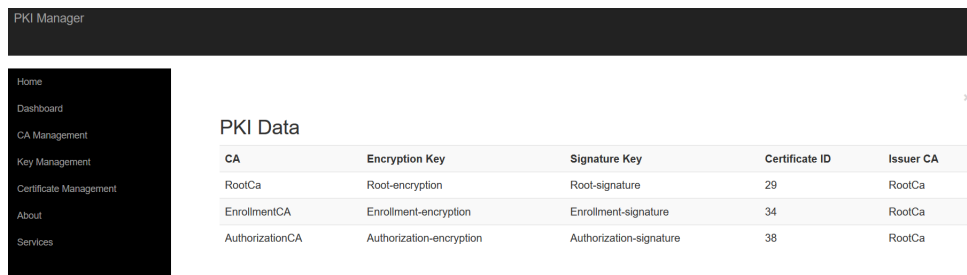
The *keys* table contains generic Key information but not the key itself, this is because the key is stored in a keystore. We only use this table to store information needed to locate the key in the keystore and view its characteristic, such as the alias, type of the key (encryption or signature), the algorithm. The foreign key *ca_id*, allows us to know which CA a specific key belongs to.

Because currently there is not a keystore that supports the storage of this type of CA certificates, we decided to store them in the database. The *certificates* table stores all the information that we need to pass to the V2X library in order to generate CA certificates. The foreign keys subject_id and *issuer_id* allow us to know which CA is the subject and issuer of a specific certificate.

3.3.3 The Backoffice
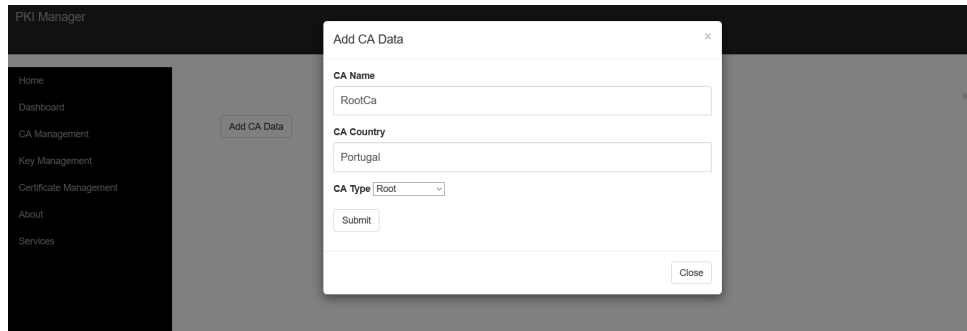
The PKI Manager is accessible through the browser only by system administrators. This backoffice application uses the V2X Library and through its services provides a GUI where administrators can visually create and configure a demo vehicular PKI. In this section we describe the functionalities and implementation details of the PKI Manager.

Upon logging in, the administrator will have the option of managing the CAs, keys, certificates, or viewing the PKI dashboard. These operations can be chosen from the index page, each having a dedicated web page. Performing a PKI operation updates the application dashboard, shown in Figure 3.11, allowing the user to understand the state of the PKI. Specifically, whether the PKI has the necessary

39

Figure 3.11: Data of the created PKI

Figure 3.12: From to add a new CA.

CAs and if each is ready to issue certificates. On the ramining of this section we take a look at the steps which an admin user has to take in order to create a PKI using our backoffice application.

CA Management   The first operation that should be done to have a functional PKI is the management of the PKI CAs. In this page, the user has the option of creating a CA by clicking a button. Performing this action displays the submit form shown in Figure 3.12. In this form the user can specify the name, country and type of CA. The last is a drop down menu which shows the already defined CA types of Root, Enrollment and Authorization. When this form is submitted, this page's Spring MVC controller at the server-side will handle the POST request by storing the CA information on the database. To be precise, our controller calls the *caManagementService* which performs the bussiness logic such as validation of the inputed information, then the service calls the caRepository that stores the information. When this logic is completed, the controller will redirect the browser to same CA management page, which makes it perform a GET request. The GET controller for this page will then handle such HTTP request by getting all the existing CAs from the database, returning them along with the requested web page to the browser. With this information it is possible to display the existing CAs as we can see from Figure 3.12, updating the table each time we get the CA management page.

The same logic is done for all the other operations, for each operation we have a spring controller which implements end-points that will handle HTTP GET and POST requests. The GET end-point has the responsibility of returning to the browser the web page with all the information needed to perform the operation. The POST end-point has the responsibility of generating resources (web pages, database information, services, etc.) on the server-side in respect to the information sent by the browser.

40
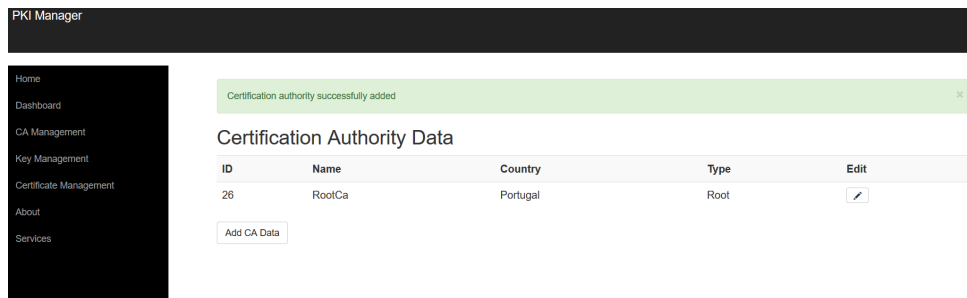
Figure 3.13: Certification authority table.
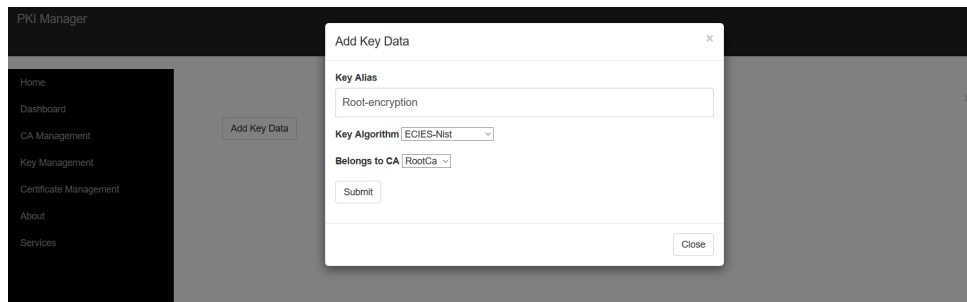
Figure 3.14: Form to add a new keypair to an existing CA

Key Management   After creating a CA, the next step is to generate its signing and encryption keypairs. Once on the Key Management page, the user is able to do so by clicking a button. This button opens a form where the key alias, algorithm and owener CA can be chosen, as we can see in Figure 3.14 . In terms of constraints for these inputs, the alias must be unique, the user must select one of the provided encryption or signing algorithms, and one of the created CAs as the owner of the keypair. Being that these selectable values are loaded by the server in the browser when it requests this page. Submitting the form will send a POST request to our server, which will generate the key given the algorithm and store it with respect to the owner CA. The generation of the keypair is done using V2X Library interface which allows the PKI Manager to access the services of this library. The storage of the keypair is done regarding two aspects, the generated keypair and its information. The keypair itself is stored on a keystore file which is loaded into our server when its execution starts. The keystore data which is inputted by the user at the client-side is stored on the database. With this method it is possible to quickly search for a key in our database using its information and then find its value on the keystore providing the alias. Upon finishing these operations, the server will refresh the Key Management page, updating the existing keys table with the new entry.

Certificate Management   The last step to have a functional CA is to generate its certificate. The Certificate Management page can be navigated to for this purpose. Once there, the user can generate a Root CA certificate or a Sub CA certificate by clicking the respective buttons. Generating a Root CA certificate involves filling the form shown in Figure 3.15 with the following information relative to the root certificate: time validity, country, and the Root CA which is the subject of the certificate. For the subject input the user is able to choose a CA from a list containing all the existing Root CAs which already

41

Figure 3.15: Form to add a certificate to an existing Root CA

Figure 3.16: Form to add a certificate to an existing Sub CA

have associated a signing keypair but no certificate. The list is provided by the server when the page is loaded and negates the user error of creating a certificate for a Root CA which is not ready to self-sign its certificate. Creating a Sub CA certificate is a very similar process with the difference that the user must fill the form displayed in Figure 3.16. In this form the user must not only specify the subject of the certificate but also issuer CA. For the certificate's subject, a Sub CA with signature keys and no certificate must be selected. The issuer must be a Root CA which is ready to issue the certificate, i.e. a Root CA which has already has a signature keypair and a certificate associated. The mapping between a certificate and its issuer will allow the PKI Manager to build certification chains in the future. When the form is submitted, our server will compile the information inputted by the user, calling the V2X Library to issue the certificate and storing it on the database. Once these operations are finished, the server will refresh the Certificate Management page, updating the existing certificates table with the new entry.

3.4 RA Service

The RA Service can be seen as the gateway which allows vehicles to request certificates from the PKI. This gives the RA the responsibility of validating the origin of the requests, making sure that each vehicle is authentic, and also validating that the destination CA is ready to receive the request. In this section we discuss the implementation of the RA service, we start by describing the technologies then we take a look at the functionalities provided by the RA.

The RA service is a module of the PKI Manager, this means that it is implemented within the same Java project and thus shares the same resources, such as the database and the internal service layer. However, to give the illusion of distance between the RA and the PKI, we decided to separate the database into the part used by the PKI Manager and the part which is used by the RA. In addition, we also separated the internal services which the PKI provides to the RA from the services used by the PKI Manager to manage the PKI. At a high level, the RA Service is a RESTful API from which the PKI Manager is able to communicate over the internet with other software programs.

A RESTful API is a set of end-points which any other application can communicate via HTTP protocol such as POST, GET, PUT, DELETE, etc. Such functions are defined in a special Spring MVC controller, each identified by an URL. In our case, we have an end-point for each service that the API exposes: vehicle configuration, enrollment, and authorization. Generally the data transferred between the client and such end-points (inside the HTTP requests) is objects from the common V2X library formatted using JSON. JSON is a human readable format for structuring and transmitting data objects, it relies on text key and value pairs. This implies that both the vehicle and server have a way of converting the original objects into a string based DTO *data transfer object* and vice versa.

To document our API we decided to use the Swagger tool. From annotations on the source code, this tool is able to automatically generate an interactive API documentation which is accessible through the browser. In this documentation it is possible to understand: what operations do the API support, what are the parameters of each operation and the return values, if authorization is needed, and even usage licence information. This allows our API to be better understood by us, and any other client software that might want to communicate with the PKI manager. In the remainder of this section we describe the implementation of each operation provided by the RAService, for each operation we take a closer look at the request, end-point, and response.

3.5   Vehicle Configuration

The Vehicle Configuration is the first operation that should be requested by the new client vehicles. This service has two main functions: to configure the vehicle within the RA, and to provide an initial configuration within the vehicle regarding the PKI. The first function allows the RA to validate the identity of the vehicle on future interactions. The last aims to prepare the vehicle to request certificates, the goal is to simulate the PKI information that needs to be installed on the vehicle at manufacture time as specified in Section 2.1.2. In a real case scenario, the RA would not have this functions, as the configuration would be done in the vehicle and RA during the vehicle manufacture. However, for demonstration and testability purposes we decided to provide this service.

The request

Figure 3.17 illustrates the configuration request which the API is expecting to receive. In order to communicate with the API through the configuration end-point, a vehicle needs to be able to build such request. The canonical key is an encoded PublicVerificationKey structure as defined in our V2X library.

```
{
  "canonicalPublicKey": "string",
  "vehicleId": "string",
  "vehicleType": 0
}
```

Figure 3.17: Vehicle configuration request in JSON format.

The id represents the vehicle's unique name (ITS identifier). Finally, the type attribute represents a specific vehicle type, e.g. ambulance, truck, etc. To transfer this information the vehicle must compile such attributes into a string based DTO which is known by the server. Such object represents the request and contains each encoded in the String type. While the vehicle type and name are already string objects, the public key needs to be transformed in order to be a part of the request. Such transformation involves the vehicle encoding the *PublicVerficationKey* using its *encode* method, then transforming the outputted byte stream into a base64 String.

The end-point

The RA service provides a POST controller dedicated to receiving the vehicle configuration requests. The first step that the RA does when it receives a configuration request is to try to save the vehicle's data in the database. However, it needs to first decode the JSON formatted data, because the request is a known DTO the RA knows how to decode it into its original objects. This is done by the *DTOtoVehicle* method that exists on the RA's service layer, this method receives the request's *VehicleDTO* and returns and object of *Vehicle* which is a database entity containing each of the DTO's decoded attributes, and thus being more useful for the business logic. To achieve this, the *DTOtoVehicle* method first decodes the base64 String canonical key to a steam of bytes and uses it to decode a *PublicVerificationKey* object. The next step is to map the vehicle type to a vehicle profile name, a profile is built for a specific type of vehicle and contains generic vehicle information such as enrollment and authorization validity information. If these operations completed without generating exceptions and the requesting vehicle was not configured before, the vehicle configuration was successful and the *DTOtoVehicle* method will return a new *Vehicle* containing the vehicle's id, profile and canonical key, which will be stored in the RA's database under the *vehicle* table as a new entry.

The response

The next step is to respond to the vehicle with the PKI information. The response built by the RA is a *ConfigResponse* DTO which always informs the vehicle of the response's origin, destination, and if the configuration was successful. Only in the positive case, the response will also contain the following PKI information: the certificate of the EA which will enrol the vehicle; the certificate of the AA that will authorize the vehicle; a list containing the certificates of all trusted AA which will allow the vehicle to trust incoming V2X messages; and the certificate of the Root CA which is the root of trust for this hierarchy. With this information the vehicle is able to request the RA for its enrollment certificate.

```
{
  "requestDestination": "string",
  "requestEncoded": "string",
  "requestOrigin": "string"
}
```

Figure 3.18: Vehicle enrollment request in JSON format.

3.5.1   Vehicle Enrollment

The vehicle enrollment operation can be used by configured vehicles to, as the name suggests, request a valid enrollment certificate. This operation involves the vehicle, the RA and the EA which is able to issue such certificate. During this communication, the RA has the function of assuring that the vehicle is configured and therefore authentic.

The request

Figure 3.18 illustrates the enrollment request which the API is expecting to receive. This DTO is composed of three elements: the destination representing the name of the EA which the vehicle is trying to contact, the encoded request which is an encoded *EnrolmenRequest* as defined in Section 3.2.2, finally the origin containing the vehicle's unique name (ITS identifier). For a vehicle to build the DTO which the RA is expecting to receive, it must be able to build and encode the *EnrolmenRequest* element of the DTO as defined in the V2X Library. To do so, the vehicle first has to generate necessary cryptographic keys. Such keys are a new verification key pair, which the public key is to be certified by the enrollment certificate, and a symmetric AES key to encrypt the request and share with the EA. With these keys, the vehicle is able to use the V2X Library in order to generate and encode the *EnrolmenRequest* data structure into a byte stream. The final step is to transform such byte stream into a base64 string and create the request DTO with the other origin and destination attributes.

The end-point

The RA service provides a POST controller dedicated to receiving the vehicle enrollment requests. The first step that the RA does once it receives such request is try to verify its origin. This is done by the verifySource method which is implemented on the service layer of the RA. This method has two main functions, verifying that the vehicle is already configured within the RA, and if the destinations EA is ready to receive the vehicle's request. To do so, this method receives the request DTO as it was sent by the vehicle. First, it gets the origin of the request and queries the RA's database for that specific vehicle. In the event that the vehicle is not found, an appropriated exception will be thrown. Then using the name of the EA on the request's destination attribute, this method tries to find the such CA on the database. If found, it checks if the CA type is set to enrollment, it has signature and encryption keys, and a certificate associated. This is done to ensure that the EA that the vehicle is trying to contact is able to issue the enrollment credential. In the case that these verifications fails, an appropriate exception will be thrown.

The last step is to try to decode the *EnrolmenRequest* from its base64 string format into an array of bytes representing the V2X Library encoded data structure.

Once the origin and destination of the request are verified by the RA, it is time to send the vehicle's request to the target EA. Since the PKI Manager and RA Service are part of the same project, this communication is achieved through a simple service call. The service layer of the PKI Manager provides the RA with the *caService* class which contains every service which the CAs are able to provide. In this case the service needed by the RA is the *validateEcRequest*, this method has the objective of validating the *EnrollmentRequest* structure, therefore ultimately verifying the vehicle's identity. In order to do so, it needs to perform the following steps: decrypting the request, validating the vehicle's signatures within, and finally returning the *EnrollmentResponse* which may contain the enrollment certificate.

The RA calls the *validateEcRequest* service by passing the *EnrolmenRequest* and name of the EA, which were found on the vehicle's request DTO; and the vehicle's canonical key and profile name found on the RA's database under the vehicle's configuration information. In order to decrypt the request, this service first gets the stored information relative to the target EA. Using the name of the EA, it gets the EA's certificate from the PKI Manager database, and its encryption/signature key pairs from the *KeyStore*. With the EA's relevant information, this service calls the V2X Library in order to decrypt the request. This is achieved through the V2X Interface which specifies the services available by the Library side to the PKI Manager. The *decryptRequest* service given the encrypted *EnrolmenRequest*, the EA's decryption key (private key) and certificate will verify if the request was encrypted to that EA, and if so decrypt it using the provided private key. To do so, it first decodes the byte encoded *EnrolmenRequest* into an *EncryptedData* structure, then using the cryptographic module of the V2X Library calculates the hash of the EA certificate and compares it to the hash that can be found within the recipient information field of the request. If there is a match, the symmetric encryption key will be decrypted using the private key of the EA, giving access to the key which encrypted the request. To decrypt the request, this method retrieves the AES cipher text field from the request and tries to decipher it using discovered symmetric key. In case that this method fails to perform such description operations or there is no match on the calculated EA certificate hash and recipient identifier within the request, an apprprieate exception will be thrown by the service. Otherwise, the signed request will be returned containing the vehicle's signatures and the requested subject attributes.

The next step that the *validateEcRequest* method does is to verify the vehicle's identity. To do this, it calls the *verifyEcRequest* service provided by the V2X Library to verify the vehicle's signatures. This method uses the recently decrypted request and the vehicle's canonical public key to first verify the request's outer signature, this assures the EA that the vehicle that build the request has possession of the private canonical key which was attributes to it during manufacture. The next step is to verify the request's inner signature. To do so, the Library gets the inner signed structure from the request, from this structure extracts the verification public key which the vehicle is trying to get certified by the enrollment credential, and finally verifies the inner signature. This assures the EA that the vehicle has possession of the corresponding private key, therefore making it impossible for the EA to issue an enrollment credential to a vehicle which does not have the correct verification key pair. The verification of the enrollment

request is only accepted if both signatures are valid. This service returns an enrollment response code which summarizes what happened during the request's validations, some possible outputs are: OK, can't parse, not the recipient, invalid signature, incomplete request, decryption failed, etc.

The response

The final step is to build the response to the vehicle, this is done by the *genEcResponse* service of the V2X Library which is called by the PKI Manager. Since it is at this step that the enrollment credential will be generated, the PKI Manager gets from its database the profile information of the vehicle using the profile name which was sent to it by the RA. This information is necessary to issue the enrollment credential as it contains information such as the default certificate time and geographic validity. The *genEcResponse* service works with the signed request; the information of the EA, including its signature key pair and certificate; the vehicle's profile information; and the enrollment response code from the last step. The first verification done by this service is with the enrollment response code, if it is a negative code such as can't decrypt, invalid signature, can't parse, etc. A new *EnrollmentResponse* 3.2.2 is built without the generation of the enrollment credential. If the response is OK, this service will issue the enrollment credential. This certificate is generated using the vehicle profile information and the inner enrollment request which contains the vehicle's verification key and requested subject attributes. Once the certificate is generated by the V2X Library, the *EnrollmentResponse* is generated including the enrollment credential. In both cases, response is signed by the EA's signature key and encrypted using the shared symmetric key. Such response is returned to the PKI Manager which saves the credential on its database marked as active, identified by its hash, and issued by the specific EA. The final step is to return the encrypted response to the RA which sends it to the requesting vehicle. To do so, the RA first makes the *EnrollmentResponse* object ready to be transferred by encoding it in base64 text and encapsulating in a response DTO. Then the RA's controller is able to return the vehicle's expected string based object. However, if the EA was not able to retrieve the shared symmetric key from within the *EnrollmentRequest*, therefore it was not able to built the encrypted *EnrollmentResponse*. In this case, the RA is notified and will send the response DTO marked as failed without the EA's response structure.

3.5.2   Vehicle Authorization

The vehicle authorization operation can be used by enrolled vehicles in order to request an authorization ticket. Such operation involves the vehicle, RA, EA, and the AA. As in the previous operations, the responsibility of the RA is to assure that the requesting vehicle is configured, and also decoding the string based request DTO so that they can be used by the PKI Manager's services.

The request

Figure 3.19 illustrates the authorization request which the API is expecting to receive. This DTO is similar to the one expected by the enrollment end-point, the only differences are the encoded request which is an *AuthorizationRequest* as defined in Section 3.2.2, and the destination references the AA

```
{
  "requestDestination": "string",
  "aRequestEncoded": "string",
  "requestOrigin": "string"
}
```

Figure 3.19: Vehicle enrollment request in JSON format.

which will authorize the vehicle. To generate and encode such a request, the vehicle is able to use the V2X Library, it just needs to generate the necessary keys first. Such keys are a new verification key pair which will be certified by the authorization ticket, and two different symmetric keys in order to share with the EA and AA. Like the previous operations, the string based DTO forces the vehicle to be able to encode such *AuthorizationRequest* in order to create the request which the RA is expecting.

The end-point

The RA service provides a POST controller dedicated to receiving the vehicle authorization requests. The first step that the RA does once it receives such request is to try to verify its origin, this is done by the same verifySource method used in the enrollment operation. However, in the authorization operation the RA processes this step differently. As explained in the authorization protocol 3.1.3, the authorization process of a vehicle is composed of multiple individual authorization requests, each for an authorization ticket. Through this first step, the RA has the responsibility of determining if the received request is the first of the vehicles' authorization process or not. The verifySource method receives the request DTO as it was sent by the vehicle, after validating that the destination AA is ready to issue certificates it validates that the origin vehicle is configured in the RA. If the vehicle is found, then this method gets the maximum number of authorization requests that the vehicle is allowed to make from its profile information. Comparing this number to the number of requests done by this vehicle (identified by the origin on the request DTO) allows the RA to know if the received request is the first of the authorization process, and therefore if the vehicle's enrollment needs to be validated, see Figure 3.3. To protect against vehicles using other vehicle's identifiers to request authorization tickets the SSL protocol is used. The communication security provided by the protocol allows to the RA to trust that the vehicles are in fact the ones that their secret identifier say they are. When a vehicle starts the authorization process it is expected by the RA that all the underlying requests will be done close to each other, in a sequence manner. However, if this is not the case it would be necessary to implement a cron based mechanism that after some time marks the vehicle so that the RA knows that its enrollment must be verified in the next authorization request regardless of the request counter. The last step done by this method is to try to decode the *AuthorizationRequest* from its base64 string format into an V2X Library data structure.

Once the RA has verified the origin, destination and determined the enrollment validation for the request, it is time to send the decoded request to the AA. To do so, the RA calls the *validateAtRequest* on the CA's service layer. From all the services which the CA's are able to provide to the RA, this

48

method can be seen as the service which the AA provides for the RA to send the vehicle authorization requests. Such method has the objective of validation the *AuthorizationRequest* structure. To do so the following steps are performed: decrypting the request, validating the vehicle's signature, requesting the verification of the vehicle's enrollment if applicable, and finally generating the *AuthorizationResponse* which may contained the issued authorization ticket.

The RA calls the *validateAtRequest* by passing the *AuthorizationRequest*, the name of the AA, the vehicle's profile name, and the flag that determines if the vehicle's enrollment is to be verified or not. The decryption of the request is similar to the process used by the vehicle enrollment operation: the *validateAtRequest* service starts by getting the AA's certificate and keys from the database and *KeyStore*. With this information it calls the same V2X Library *decryptRequest* service which returns the signed request structure. With the decrypted request the AA is able to verify the vehicle's signature using the to be certified public verification key in the request. Such validation assures the AA that the vehicle which created the request has possession of the corresponding private key. In the eventuality that request is not valid because, for example, the request is badly formed, it was not encrypted to this AA, the signature is invalid, etc the response code to be delivered to the vehicle in the response will be of error.

The response

If the response code is positive and the enrollment validation flag was set to true by the RA in the last step, the next step is to request the EA for the validation of the vehicle's enrollment. This is done by first extracting the *AuthorizationValidationRequest* structure from the now decrypted *AuthorizationRequest*, the *AuthorizationValidationRequest* structure contains the EA encrypted vehicle's enrollment signature and the shared request. The AA is able to find the hash of the EA's certificate within the shared request. With this information it is able to get the name of the EA from the database and send it the *AuthorizationValidationRequest* by calling the *validateEnrollment* CA service. In the case that the RA determined that the request is not to be enrollment verified, the call to the EA's *validateEnrollment* service is skipped,

The first step that this EA service performs is the decryption of the received request, to do so, it relies on the name of the EA to get its certificate and decryption key from the persistent layer of the PKI Manager. Once the decryption parameters are set, it calls the V2X library *decryptRequest* service to decrypt the request. The next step is for the EA to validate the vehicle's enrollment by verifying its enrollment certificate and signature. First, this service gets the vehicle's credential from the database (stored by the EA at the enrollment phase) and checks if it is still marked as valid, if this is the case, then the service cryptographically validates the enrollment credential using the *verifyCertificate* service provided by the V2X Library. This service, given a certificate and the corresponding certification chain will verify if such certificate belongs in that chain by decrypting its signature using the public verification key of the next certificate in the chain, in addition to this, it also validates the expiration date on the certificate. The return value is a simple boolean which, if true, allows the EA to continue with the vehicle's enrollment validation by decrypting its enrollment signature. This process can be started by calling the V2X Library *verifyEnrollmentSignature* service which given the vehicle's enrollment credential

and *AuthorizationValidationRequest* is able to extract the enrollment signature from the request and verify it using the credential's public key. If the signature is valid the EA has validated the vehicle's enrollment and returns a positive response to the AA. In the case either the credential or the signature is not valid, a negative response will be sent yo the AA.

The final step is to build the response to the vehicle which may contain the requested authorization ticket. To generate such response the AA, first gets its signature key pair; its certificate; the vehicle's profile information; the signed request; and the response code which resulted from the validation of such request and, if applicable, the vehicle's enrollment. With this information, the AA calls the *genAtResponse* V2X service which is the response code is positive will issue the authorization ticket using the AA's certificate, private signature key and vehicle's profile information, including it in the response. The response structure is generated always signed by the AA and encrypted to the vehicle, (using the same symmetric key from the request). The final step is to return this response to the RA marked as positive so that it can be returned to the vehicle.

3.6 Vehicle Manager

The Vehicle Manager has the responsibility of generating client vehicles for the PKI Manager, at a high level we can see the Vehicle Manager as a vehicle manufacturer which is trusted by the RA service. In addition to generating the vehicles, it is this component's responsabilty to provide a V2X environment where the generated vehicles are able to communicate with each other. To ease the testability of such environment, the Vehicle Manager reads the application properties to load the execution variables, such as the number of vehicles, number of times a AT is used, pattern of certificate usage, etc.

At this point we have a REST API which is documented using the Swagger tool, Although the automatic and interactive documentation is useful in its own way, the main reason why we chose Swagger to document our the RA Service is the possibility of automatically generating client libraries for the API. For this purpose, we used the *Swagger Codegen* tool, an open source project available on the *GitHub* repository. This tool works with the URL of our API's documentation and is able to generate a client project where the communication layer is made to specifically communicate with the API. This allowed us to save time abstracting from the communication details and focus on the functionality of the Vehicle Manager.

The Vehicle Manager is a simple multi-threaded *Java* project which can be started using the *Main* method. The first operation done by the this method is to initialize the generation of vehicles, which is managed by other class. The output of the vehicle generator is a list containing the vehicles, each with a fixed size random *String* as the identifier, and one of the V2X library supported encryption/signature algorithms. At this point the main method knows how many vehicles will participate in the execution so it populates each vehicle with a list of its neighbours.

The multi-thread attribute of the Vehicle Manager comes into play right after the generation of the vehicles, where each vehicle represents a different thread in the execution of the application. Our objective with this is to make the V2X communications seem more realistic where each vehicle is sending

messages to its peers in no particular order, resutling in different communication situations.

The execution of the threads starts when the main method calls the *start* method of the generated vehicles. When started, each vehicle performs the protocol described in Section 3.1.3 by requesting the operations of the RA Service of vehicle configuration, enrollment and authorization. Using the V2X Library enables each vehicle to generate the most complex request elements. The result of the protocol is the installation of the CA certificates, the enrollment credential and a list of authorization tickets within each vehicle.

Once a vehicle has all the needed certificates it is time to start communicating with its peers. To do so, it starts two timers: the *CAMtimer* and the includeCertTimer. The first has the responsability of periodically sending CAMs, each time electing an authorization ticket to sign the message (currently used AT). The second timer informs the vehicle when it is time to send the full authorization ticket within the signer information of its next message so that it can be distributed.

The process of broadcasting a CAM is composed of three steps: checking for unknown authorization tickets, generating the message, and sending it. In the first step the vehicle searches its internal list of unknown authorization tickets, if any hash is found then it will be included in the next CAM so that the corresponding full certificate can be requested. The generation of the CAM itself is done using the V2X Library with respect to the currently used AT, the corresponding signature keys, the list of unknown ATs, and wether to include the full signing AT or its hashed reference. Finally, in order to send the CAM the vehicle simply iterate through the list of neighbour vehicles calling their respective *receiveCAM* method.

When a vehicle receives a CAM the first step is to look for the hash of its currently used AT within the list of requested certificates. If found, then the vehicle sets a variable to include its full AT on the next message. This mechanism allows the distribution of the currently used AT in the cases that other vehicles have requested it. The next step is to try to verify the message. For this purpose, the vehicle first reads the message signer identifier in onder to understand the type of signature, i.e. if the message includes the signer's AT or the hash value.

In the first case, the vehicle verifies the signer's AT against the certificates of the trusted authorization authorities. If the AT is signed by a trusted authorization authority, the vehicle stores it as trusted in a hash map enabling access on future communications, and proceeds to verify the message. This is a simple process where the signature is verified against the signer's AT, and the message generation time is within the certificate's expiration period.

In the case that the vehicle receives a CAM containing the hashed AT, the vehicle starts the verification of the message by looking for that digest on its trusted AT hashmap. If the hash is found it means that the AT was already verified and roots to a trusted AA certificate, so the message is able to be verified using the same method as in the previous case. However, if the AT is unknown the vehicle adds the digest to the list of unknown ATs so it can be requested in the next message. Because the message was not able to be verified due to the missing AT, it is stored in a list of pending messages which await verification until the full signing AT is included within an incoming CAM.

When a vehicle receives a CAM containing the full certificate it can be the consequence of this vehicle's request for an unknown certificate. Therefore, the vehicle tries to verify both the new and

pending messages. For each pending CAM we compare the digest on the signer identifier to the digest of the received AT. A match means that the pending message was signed by that same AT, which if trusted by the vehicle, enabling the verification of the message.

3.7 Summary

In this chapter we presented our solution to the problem of implementing a PKI solution that allows a V2X environment while preserving the privacy of the vehicles. We started by overviewing the architecture of the solution, learning about its main components: PKI Manager, V2X Library and Vehicle Manager. We studied how such components are organized in a client-server architecture, assuming the PKI Manager as the server and the Vehicle Manager as the client. In addition, we seen that these two components communicate through the API of the server, the RA Service and utilize the V2X Library in order to generate the necessary messages. We concluded this chapter by presenting the detailed implementation of each component, where we also explained the technologies and external libraries adopted in order to implement our components. In the next chapter we perform an experimental evaluation to our V2X system, where we test the performance of the server, the resource usage of the client and finalize by discussing the security and privacy properties achieved.

Chapter 4

Evaluation

This chapter describes the evaluation of our V2X system, which comprises the PKI Manager and Vehicle Manager. The three main non-functional requirements of our V2X system are performance, security, and privacy. With this in mind, we intend to answer the following questions:

1. Since interoperability is a constant concern, does the system provide acceptable performance?

2. Does the system deliver the necessary conditions for vehicle privacy, authentication and overall security?

3. Does the implemented V2X environment provide vehicle privacy at an acceptable cost?

The following sections address these three questions, staring with the performance and resource usage tests, moving to the privacy and security concerns.

4.1   Performance

Performance is a fundamental concern when testing systems that address a sensitive subject such as road safety. The most common metrics of performance are latency and throughput: latency is the time necessary to complete certain operations and throughput in our case is defined by the number of requests that the PKI Manager processes in a determined time frame. While latency gives us an idea on how fast users are able to use each service, throughput relates to the capacity of the PKI Manager to process multiple requests in a given amount of time.

Such two performance metrics correspond to the metrics measured by our tests to the PKI Manager at different variants. To perform this evaluation several tests were done regarding the backoffice application, and the interaction between the Vehicle Manager and PKI Manager. The environment in which they execute is the following:

- A single installation of the PKI Manager application as the server.

- A basic but functional vehicular PKI including one Root CA, one Enrollment CA, and one Authorization CA.

- A single installation of the Vehicle Manager application as the client.

Our goal with this setup is to provide a simple but complete V2X environment, where we can use the installation of the PKI Manager to test the performance of the backoffice and RA Service, and use the Vehicle Manager to execute tests focused the resource usage of V2X communications.

4.1.1 PKI Manager

The PKI Manager is the component of our system which comprises the web application. Therefore, the performance tests done to the PKI Manager have the main goal of understanding how the application behaves under different work load conditions. These tests focus on two aspects: latency measurements of the operations supported by the backoffice application, and the latency resulting from the communications between client vehicles and the PKI through the RA Service operations.

Because the PKI Manager tests involve client-server communication over the network, we executed them on two different machines connected through the same network. The server ran on an Asus laptop with 16GB of RAM and an i7 Intel CPU. For the backoffice testes, the URL was accessed on the browser of the second machine and the measurements were taken using the network tab on the browser's developers console. Regarding the tests to the RA Service, the client vehicles were generated on the second machine using *Apache Jmeter* [33], an open source software designed for testing the performance of web applications. This configuration allows us in both test suits to consider the latency introduced by the network communications and the latency of the processing time at the server.

Backoffice Application

The operations performed by the administrator application are simple but essential in order to have a functional PKI to serve as the backbone of trust within V2X communications. The processing is mostly located at the V2X Library, creating a waiting period in the backoffice application. On the administrator side, it makes sense to only test the latency of the operations performed by one user only, as this application is not meant to be used concurrently by a large number of users. Next, we take a look over the operations evaluated on the backoffice: the generation of keys and certificates.

The operation to add a key involves the following steps: generating a cryptographic key using the V2X Library, then storing it on a keystore and its information on the database. The operation to add a certificate is more complex than the previous, as it also involves searching for the keys which will be relevant to issue the certificate. Because of this factor, we included on the add certificate test the number of keys that exist on the system to provide a comparing factor amongst the different latency times. For each operation the timer started when the form was submitted (POST) and stopped when the page finished loading (GET). The reasoning behind not considering the generation of CAs in the test suit comes from the fact that, unlike the tested operations, generating a CA consists only in creating a simple database object and storing it.

In Table 4.1 we can observe the time necessary to complete the operations involving our backoffice application, which preclude user interaction. Overall from the perspective of an administrator the results

Operation	$Time$	$Keys$	$Standard Deviation$
Add key	71.6ms	1	14ms
Add certificate	73.7ms	4	10ms
Add certificate	81ms	20	13.5ms
Add certificate	103ms	30	14ms
Add certificate	128.6ms	40	16ms
Add certificate	178.7ms	60	19.5ms

Table 4.1: Time needed to add a certificate and a key, calculated over twenty measurements.

are reasonable and will not negatively affect the usability of the application. Regarding the operation to add a certificate, the operations for adding a single key or certificate are fast, taking on average 71.6 and 73.7 milliseconds. There is a linear relation between the time that the system takes to add a certificate and the number of keys that exist on storage. This happens due to the nature of the operation, in which the server has to query the database and the KeyStore in order to find the cryptographic keys of the subject and issuer before calling the V2X Library to issue the certificate, and finally storing it on the database. This increase in delay is not critical, as considering a realistic number of keys (i.e., in the order of a few dozen keys) it does not affect the user experience to a point that it is noticeable.

RA Service

The RA Service API is the gateway from which the vehicles are able to interact with the PKI to request certificates. Comparing with the backoffice application, the operations done by the RA Service are more complex and subject to more load due to the possibility of concurrent requests from the client vehicles. Therefore, the tests done to this component are designed to evaluate the behaviour of the API at different levels of server load. Ideally, to ensure that the results are as close to the reality as possible, API load tests are run on a production or equivalent system. However, in our case this was not possible so the results may differ depending on the machine which they run. In the remaining part of this section we take a look at the performance tests performed to the RA Service starting with latency of the supported operations moving to the throughput measurements.

Latency Figure 4.1 shows the tests done to the RA Service: latency resulting from the configuration, enrollment and authorization of vehicles respectively. As discussed before, we used *Jmeter* in order to test each end-point. This tool allowed us to configure different number of users or virtual threads and send concurrent requests to the server automatically. For each operation we performed a total of 50 tests across different thresholds of concurrent users. We started with one user, tested 10 times, then increasing the number of users in order to increase the load on the server and provide a comparing factor between the latency times. The plot for each operation relates the number of concurrent users to the time needed to perform the operation. The time is the average of 10 measurements, where each test was performed when the previous was done processing. For each test, *Jmeter* started the timer when the POST request was submitted by the thread and stopped it once the response was received.

(a) Vehicle Configuration

(b) Vehicle Enrollment

(c) Vehicle Authorization w/o enrollment validation
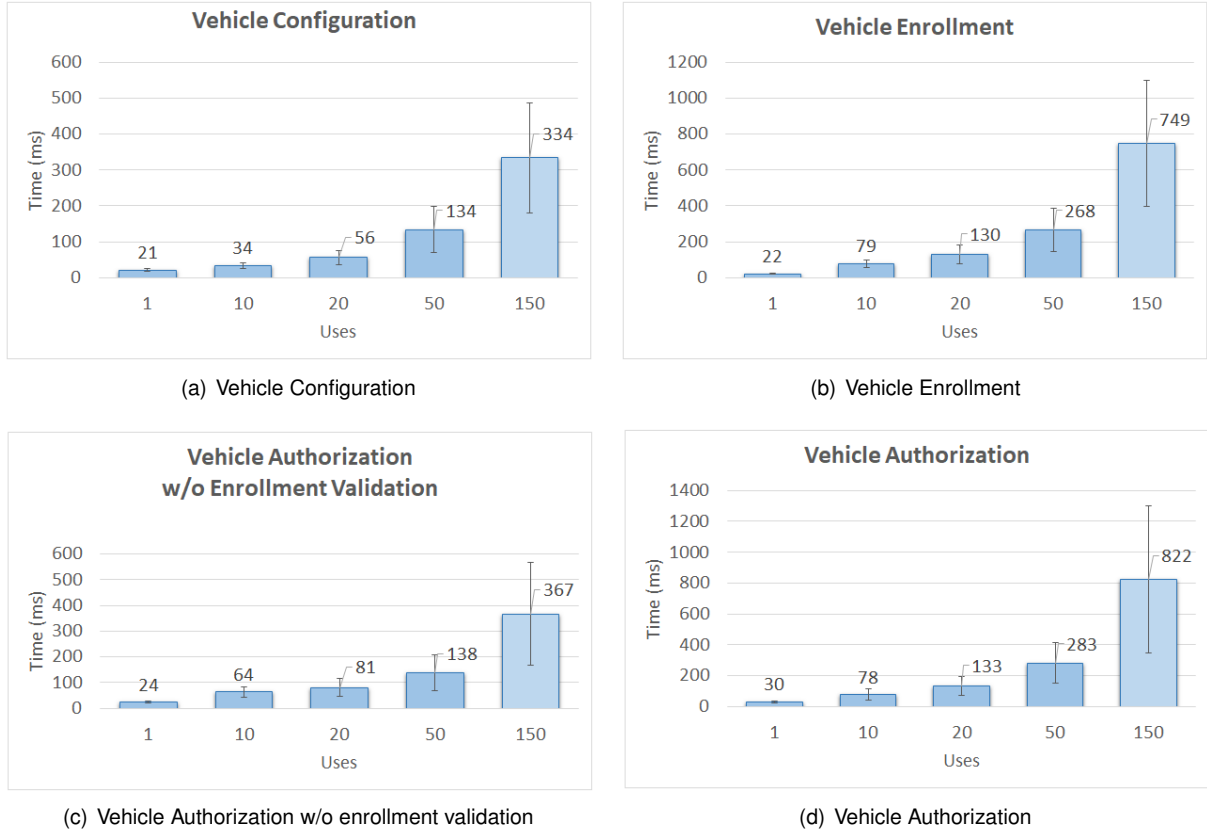

(d) Vehicle Authorization

Figure 4.1: Latency measurements for the RA Service operations

Meaning that the processing time done at the client side to build the requests and verify the responses is not included in our test results. All three operations involve reading the database, and the V2X Library to decode the request and build a response. In addition to this, both the enrollment and authorization of vehicles use the library to validate the vehicle's request and issue the certificate. For the authorization end-point we tested both use cases, as represented by Figure 3.3: the authorization process without the enrollment verification (c), and the full authorization process (d).

As we can see from the plot (d), the authorization of vehicles while verifying their enrollment is the operation which overall introduces more latency, with values varying from 30ms to 822ms. This happens due to the fact this operation is the most complex in terms of computation. Specifically, in the request validation where this version of the vehicle authorization requires the validation of two signatures: the signature that proves the possession of the verification keypair, and the vehicle's enrollment signature. In addition to this, the vehicle's enrollment credential is also verified.

The second most expensive operation is the enrollment of vehicles (b) with latency values ranging between 22ms and 749ms. Comparing to the previous operation the enrollment of vehicles is simpler. The main difference comes from the validation of the vehicle's request, which in this case only requires the validation of two signatures: the canonical signature and the proof of possession signature. However, as we can see from plots (b) and (d) for all numbers of users the enrollment operation introduces almost as much latency as the more complex authorization, with values between 30ms and 822ms. This is because, unlike the previous operation, the enrollment of vehicles also requires saving on the database

56

each enrollment credential issued, which introduces a bottleneck on the enrollment operation.

Perhaps the most interesting comparison allowed by these tests is the comparison of the latency introduced by the two different vehicle authorization methods. As we can see from plots (c) and (d), the authorization without vehicle configuration introduced by our RA Service is on average faster than the standard full authorization flow. For example, considering the lowest server load of 1 user the new authorization flow represents an improvement of 16.6% of the full authorization. As the number of concurrent users increases so does the improvement reaching its maximum value on the 150 user threshold with an improvement of 55.7%. Since this version of the authorization is the most frequently used throughout the authorization process of the vehicles, the performance gains mean that the efforts presented in Section 3.1.3 to optimize the authorization process as a whole were successful from a performance point of view.

The configuration of vehicles (a), being the simplest operation, introduced the least latency reaching a maximum of 334ms. Unlike the previous operations, the configuration of vehicles does not require signature validation at the application level or issuing certificates. It mainly requires decoding the request from the vehicle and read/writes to the database. However, the configuration of vehicles shares the same problem as the enrollment operation as the number of concurrent requests increases, i.e., database inserts become more and more expensive. At a 150 user mark it becomes almost as slow as the authorization of vehicles w/o enrollment validation (c), which is an operation that in terms of business logic is more complex.

One way to solve the problem shared by the configuration and enrollment operations is to increase the scalability of the data access layer by effectively managing the database writes. This can be achieved by using batch calls instead of single insert calls. Since we are using the default single insert calls, the number of writes results in same number of SQL *insert* operations, which in turn result in that equal number of database round rips. Batching is a mechanism capable of grouping the *inserts*, reducing the database round trips, and consequently increasing the performance of the data access layer.

Throughput   To test the throughput of the PKI Manager we used *Jmeter* to generate requests to the vehicle authorization end-point since it is the most complex operation. This tool allowed us to configure five tests each with different ramp-up times. Each test was configured to 600 threads (users). The ramp-up period for each test was 60, 40, 30, 10 and 5 seconds respectively. Ramp-up period indicates the time duration that *Jmeter* takes to start all the threads for that test. For example, if the ramp-up time is 10 seconds in a test with 10 threads it means that each thread will start after (10/10) 1 second. Having this mechanism in mind we reduced the ramp-up time in each test while maintaining the same number of users to increase of concurrency and work load on the server, providing a comparing factor between the resulting throughput measurements. It is important to note that, as explained in the documentation of *Jmeter* [34], the throughput is calculated as (number of requests) / (total time) where time is calculated from the start of the first thread to the end of the last. For our tests the number of requests is constantly 600 and the total time is the period of time which took the test to fully execute. Specifically, starting with the first request (thread 1) and ending when the response to the last request is received (thread 600).

Ramp-up	$Latency$	$LatencyStandardDeviation$	$throughput$	$TestTime$
60s	29ms	7.2ms	10/sec	60s
40s	28ms	5.9ms	14.9/sec	40s
30s	34ms	5.6ms	20/sec	30s
10s	592ms	496.6ms	52.8/sec	11s
5s	2659ms	1480.69ms	62.5/sec	9s

Table 4.2: Throughput measurements to the authorize vehicle operation with 600 threads.

Table 4.2 relates the throughput measurements for each test done to the authorization of vehicles. In this table we can see the average latency of the 600 requests, the standard deviation of the latency, the throughput value, the percentage of failed requests and the test execution time. As we can see from the results, for the first three tests the server responded very well to the increase in the work load. We can see in the first test that for a ramp-up time of 60s the throughput was 10 requests per second. As the test was running for approximately 60 seconds, we can conclude that all the 600 requests were completed in approximately the same time as the ramp-up period meaning that the server was not delaying requests. This factor along with the low average request latency of 29ms, low standard deviation of 7.2ms and request error rate of 0% allows us to conclude that this test was not able to reach the full capacity of the server. Therefore, the throughput was not higher because of the high ramp-up time that took to create the 600 requests. The same trend was maintained for two more tests, in the test with 30s ramp-up we can see that for twice the initial work load, the server responded with double throughput of 20 requests per second. This, along with the request latency still being low and no request errors is a further indicator the server was not yet saturated.

However, once we increase the load by setting the ramp-up time to 10s the average latency of the requests started to increase to 592ms, which resulted in a higher standard deviation of 496.6ms as some requests were being processed fast while other were being delayed. For this test the throughput of the server was 52.8 request per second meaning that it was no longer increasing linearly in function to the increase in the work load. This is a sign that we are closer to reaching the server's capacity. Despite this, the average latency is still acceptable and because the error rate is 0 % this test qualifies as a good performance output. As expected in the next test where we set the ramp-up to 5s, corresponding to twice the load as in the previous test, the throughput while still high (62.5/sec) increased at a lower rate comparing to the last 4 tests. The high average request latency of 2659ms and 12% error rate on the requests is an indicator that we reached the full capacity of the server and for this work load it does not provide good performance.

4.1.2 Vehicle Manager

At the Vehicle Manager level, the focus of the latency/throughput metrics tests shifts from performance to resource usage. This is because the performance of the V2X communications could not be accurately measured by only using the Vehicle Manager. In the previous cases where the performance of the web application was tested we were able to relate the number of users to the latency needed to complete

Certificate	$Size$	$Quantity$	$TotalSize$
Root CA	272 bytes	20	5.3KB
Enrollment CA	288 bytes	100	28.1KB
Authorization CA	288 bytes	100	28.1KB
Enrollment Credential	208 bytes	1	0.2 KB
Authorization Ticket	200 bytes	1825	356.4KB
AT private key	32 bytes	1825	57KB
			475.1KB

Table 4.3: Estimated vehicle certificate storage demand

certain operations, which gave a general idea of how the system reacts under different work loads. This is not the case with the Vehicle Manager. The time needed to send messages from one vehicle to another (represented by objects within the Vehicle Manager) depends entirely on the system where the application is running. Without access to vehicle specialized V2X hardware the time taken by our testbed computer to generate, send and validate V2X messages is not conclusive. In addition to this, the network latency resulting from sending messages could not be accurately simulated because the communications within the Vehicle Manager's vehicles are represented by simple method calls.

In order to have a general notion of the possible quantities and the necessary storage capacity, Table 4.4 lists the type of certificates, their size, and possible quantity. As described in Section 2.1 all vehicles have to store different certificates on their local system. To estimate the size of the different certificates we used the *java.lang.instrumentation* package [35], where we measured size of the ASN.1 COER encoded certificates (i.e. encoded using the V2X Library methods). Regarding the possible quantity of ATs stored within the vehicles, the values are estimated considering the AT usage pattern presented in Section 2.1.4, assuming that a vehicle uses on average five certificates per day with a refill time of one year. In terms of numbers, the expected AT number can be calculated as (5 certificates) * (365 days) which equals to 1825 ATs. For the number of CA certificates we assumed a decently sized truststore installed on the vehicle. As we can see from Table 4.4, the authorization tickets and their corresponding private keys are the elements which demand more storage space (413.4KB), corresponding to approximately 87% of total. It is also important to note that the number of the certificates can change from vehicle to vehicle depending on the usage pattern and the desired level of privacy, so the overall certificate storage demand may not be the same for different types of vehicles. This is further discussed in the following section.

4.2 Security

In this section we analyse how our system achieves the security goals previously defined in Section 1.3. In particular, we discuss the following security properties: confidentiality, data authenticity, and authorization.

4.2.1 Confidentiality

Confidentiality over the communications in an important characteristic of our system. Given that it falls outside the scope of V2X communications to restrict access to messages to certain users, the confidentiality aspect of our system only applies to the communication between the Vehicle Manager and PKI Manager.

To ensure confidentiality, all the information transmitted from and to the client is sent through an SSL secure channel. In this communication channel the Vehicle Manager is authenticated, where the server requires its certificate in order to accept communications.

In addition, we also provide application level encryption where every certificate request is encrypted by the vehicle so that only the intended recipient CA is able to decrypt it. This feature combined with the encryption of the response to the original vehicle allows the confidentiality of the vehicle's identity, preventing its disclosure during the certificate requests.

4.2.2 Data Authenticity

Data authenticity enables the receiver of a message to confirm the origin and integrity of the data. In our system, this property applies to both the client-server communication and V2X messages.

Regarding the Vehicle to PKI communication, the authenticity of the messages is achieved at two levels: first at the channel and then at the application levels. The sender of messages is the Vehicle Manager and the recipient is the PKI Manager. When the vehicles are manufactured, the RA trusts the new connections because of the SSL connection with the Vehicle Manager, which can be seen as the vehicle manufacturer. However, after the configuration of such vehicles the authenticity of their certificate requests is achieved at the application level, through the use of digital signatures.

The authenticity of the V2X messages is a similar process where a vehicle being the recipient of a V2X message is able to trust the authenticity of the message by verifying the signature of the sender vehicle against a trusted certificate chain.

4.2.3 Authorization and Authentication

Authorization over services requires user authentication. In our system this is applied in two aspects: the authentication of the administrator to manage the PKI and the client vehicles to request certificates.

The authentication of the administrator is accomplished through the use of a username/password mechanism, where the credentials are stored within the database of the server. Specifically, the password is hashed with the *Bcrypt* algorithm before being saved to avoid mainlining it in clear text. The implementation of the hashing function is provided by *Spring Security* and is prepared to internally generate and maintain a random salt value to be hashed alongside the password.

The access to the data stored within the database is protected by username/password authentication. In addition to this, the access privileges are different depending on the type of user accessing it.

Regarding the authorization of the vehicles, this is achieved through the RA service of vehicle configuration. Only vehicles which completed this service are recognized by our server and are able to request

certificates.

4.3 Privacy

In our system preserving the privacy of users is one of the most important concerns. Losing privacy might mean that an attacker is able to track vehicles using the V2X communications, which is undesired. The main system variables that might affect the privacy of its users are the number of pseudonym certificates, and the usage pattern. The more pseudonym certificates a vehicle has available, the better is the privacy because the vehicle will have more choices when changing pseudonym. However, as described in the previous sections the increase of privacy comes with the overhead of the increase in certificate storage demand, number of individual certificate requests, and consequently processing at the vehicle. In this section we analyse different V2X scenarios, where we relate a vehicle's movement pattern, the pseudonym usage, possible types of attackers and the resulting level of privacy.

4.3.1 V2X Scenarios

Regarding the types of vehicle trips, we wanted to choose real life cases where V2X communications could be involved. For the privacy scenarios we considered the following cases:

- **Vehicle A:** Round trip, the typical daily home to work commute.

- **Vehicle B:** One-way Trip, from point A to B.

The usage of certificates to sign V2X messages introduces privacy concerns over the communication. Therefore, the pattern which vehicles use pseudonyms and the frequency in which the certificate are changed are very impactful variables. We considered the following pattens:

- **Pattern A:** The vehicle always uses different pseudonyms.

- **Pattern B:** For a period of time the vehicle uses all the available pseudonyms then, if necessary, reuses the certificates in a different order.

To represent a challenge to the system's privacy, we choose different types of attackers differentiated by their power to capture messages within an area of operation.

- **Attacker A:** Controls a single section on the road while being able to visualize the victim vehicle and capture all messages.

- **Attacker B:** Controls two distant sections on the road where the victim vehicle will pass.

- **Attacker C:** Attacker is moving for a limited period of time following the victim vehicle.

4.3.2 Privacy Levels

Table 4.4 organizes the scenarios from the lowest possible privacy to the highest.

Vehicle Type	$Certificate Pattern$	$Vulnerable To Attacker$	$Privacy Level$
A	B	A,B,C	Low
B	B	B,C	Medium
A	A	C	High
B	A	C	High

Table 4.4: Privacy scenarios ordered by level of privacy.

Low Privacy

Starting with the least privacy, vehicles of type A (round trips), using and reusing certificates as in th pattern B with low pseudonym change (e.g. using the same certificate for hours) are vulnerable to all types of attackers. Considering the attacker A there is a window of vulnerability if the vehicle passes the area controlled by the attacker twice with the same pseudonym. In this case the attacker is able to map that pseudonym to the victim's vehicle. Eventually, when the vehicle reuses the compromised pseudonym it may be possible for the attacker, if capturing the messages, to infer the vehicle's route. The same situation happens when considering the attacker B but with a bigger window of opportunity as the vehicle passes the areas controlled by the attacker twice as more. Because the frequency of certificate change is low it directly affects the privacy of the vehicle when considering the attacker C. In this case, the longer the vehicle uses the same certificate the bigger the probability of the attacker seeing that constant message broadcast by the vehicle, therefore mapping the used pseudonym to the identity of the vehicle and enabling short-term tracking. Despite the fact that this pseudonym usage pattern is worst in terms of privacy for the vehicles usually commuting, it is the least expensive in terms of certificate storage demand because the vehicle is reusing certificates with a low change frequency.

A privacy improvement for the same conditions (Vehicle type A, and certificate pattern B) would be to increase the frequency in which the vehicle changes pseudonyms. In this case, the vehicle would be more protected against the three types of attackers. In the case of attacker A and B, this is achieved because it would reduce the risk of the vehicle still be using the same certificate while passing through the zone(s) controlled by the attacker. The increase in privacy against attacker C is based on the unlinkability attribute of the pseudonym certificates. This means that when a vehicle changes pseudonym the new certificate can not be mapped to the old, therefore reducing the ability of the attacker to infer which certificate belongs to the vehicle. However, in order to assure the unlinkability of the pseudonym certificates care must be taken regarding the aggregation of information written on the certificate. For example, if a set of the vehicle's certificates have the exact same expiration date an attacker could infer that they belong to the same vehicle, disrupting the unlikability of the affected certificates.

Medium Privacy

Medium privacy is achieved for vehicles of type B (one-way trips) using the certificate pattern B. In this case the vehicle can expect an improvement in the privacy level because it does not pass the same road twice. This is particularly effective when considering attackers of type A and B which control fixed

areas of such road. For the first type of attacker, it is only possible to map a certificate to the vehicle if the vehicle passes the controlled area twice on different trips using the same certificate. Regarding the attacker B, it is possible to compromise a certificate if the vehicles passes the two controlled areas using the same certificate. As in the previous case, increasing the certificate change frequency will further decrease the power of the three attacker to compromise certificates for when the vehicle reuses them.

High Privacy

The best privacy for all vehicle types is achieved through the usage the certificate pattern A. Because certificates are not reused, even if the attacker manages to map a pseudonym to the victim's vehicle it is only possible to track the vehicle for the duration that it uses that certificate (short-term tracking). Increasing the certificate change frequency will further reduce the power of the attacker.

4.3.3 Discussion

A general rule of thumb that is important to consider when using pseudonyms is that a pseudonym is only effective in a crowd. In vehicular communications this means that a pseudonym certificate is only effective if there are more vehicles around using them. If a vehicle is isolated on a road sending V2X messages signed with a pseudonym certificate then that certificate can be immediately mapped to the sender, even considering a high pseudonym change frequency.

We have seen in the scenarios presented above that pseudonyms are particularly vulnerable to reuse attacks where the attacker is able to map a specific certificate to a vehicle. In this case the pseudonym becomes compromised because the attacker will be able to link that certificate to the vehicle when it reuses it. The more certificates are compromised the bigger the chance of the attacker to infer a vehicle's route by observing the compromised certificates being reused at different locations. However, assuming that vehicles carry thousands of pseudonyms, a few compromised certificates might not be conclusive for the attacker. It all depends on the power of the attacker to compromise certificates by controlling areas and/or following the victim vehicle, the number of certificates reused by the vehicle, and the randomness in which they are selected.

We also have seen that having a low pseudonym change frequency is particularly vulnerable to short-term tracking attacks where the attacker is following the vehicle and is able to: (a) see the vehicle using the certificate while isolated, or (b) infer it by the captured messages which are signed constantly by the same certificate. We concluded that the best way to preserve privacy is to increase the number of pseudonyms, which enables the possibility of a higher pseudonym change frequency and lowers the need to reuse certificates.

More concretely, the price in terms of storage in order to have acceptable privacy depends on the travel distance of the vehicles. In the previous section, we estimated that if a vehicle uses on average 5 pseudonyms in a day with a refill time of one year it would need approximately 486.6KB of certificate storage space. Depending on the distance that a vehicle travels each day, it may need to use more that 5 certificates resulting in the reusage of certificates, and privacy loss in most cases. To assure that this

does not happen a vehicle that spends more time on the road needs more certificates. For example, assuming a vehicle changes certificate each 10 minutes and an average travel time of 2 hours each day. The vehicle would need approximately 12 certificates a day in a total of 4380 for a year. Assuming the same number of PKI certificates as in the previous section 4.1.2, this number of pseudonyms would increase the storage demand to approximately 1MB.

4.4   Sumary

In this chapter we presented the evaluation results of the developed V2X system. We started by evaluating the performance of the PKI Manager, then looked at the resource usage of the Vehicle Manager and finished by discussing the security properties and privacy of the system.

The performance of the PKI Manager was measured at two different variants: the backoffice application and the RA Service. In the evaluation of the backoffice we measured the latency of the most complex PKI operations, the generation of keys and CA certificates. We concluded from the results that the measurements were reasonable, considering the complexity of the operations, and would not negatively affect the usability of the application by an administrator. From the results it was possible to conclude that the most expensive operations in terms of latency are the full authorization and enrollment of vehicles. The new authorization of vehicles introduced by our RA Service was fastar than the standard full process.

Lastly we presented how our system provides the security properties of confidentiality, data authenticity and authorization in both vehicle to PKI and V2X Communications. We concluded this chapter by analyzing how the privacy of the vehicles is maintained in V2X communications, discussing the different levels of privacy achieved by different vehicle types and pseudonym usage patterns.

Chapter 5

Conclusions

In this thesis we presented a solution for a system that incorporates V2X communications, a vehicular PKI and a RA. Next, we summarize our main achievements and discuss some future work.

5.1 Achievements

In the work, we developed a V2X ecosystem which is compatible with the most recent standardization efforts done by ETSI. Starting with the PKI Manager containing the vehicular PKI and the RA Service, which vehicles to interact with the PKI in conformance with the standardized certificate request formats. Going to the Vehicle Manager representing the role of a vehicle manufacturer, generating client vehicles an allowing V2X communication within its functionalities.

The developed system achieves a first approach to identity management of the vehicles on the road while preserving their privacy. The PKI Manager provides a way of analyzing the necessary certificates and messages, allowing the measurement of the size of such data structures and the impact that they might have on the vehicles who store them. The Vehicle Manager provides a method of seeing the V2X communications in action, allowing us to analize the privacy of the communication by looking at the received and sent messages for each participant vehicle.

Our contribution with the RA Service manages to improve the Vehicle to PKI interaction by serving as a proxy capable of configuring new vehicles and authorizing them for future communications. According to the results, the RA Service also manages to improve the performance of the vehicle authorization process as a whole.

In the end the results from the performed tests are as expected. Regarding the security, the used technologies and protocols follow the commonly recommended security practices. However, the security of the vehicle configuration, which represents the first interaction between the Vehicle Manager and PKI Manager, could be further improved as of now it only depends on the SSL protocol. The privacy of V2X communications as an integral part of the system's security cannot be optimized by choosing a fixed strategy, care must be taken in order to understand the vehicle's routines and finding a compromise between the privacy its associated overheads.

The performance of the PKI Manager is also something to take into consideration. Overall the results are acceptable considering the complexity of the operations. However, when increasing the work load on the server by increasing the number of concurrent users, the operations started to take considerable more time to be performed and in some cases errors where introduced. While this is problem for this particular proof of concept system, it can be easily corrected by introducing a distributed infrastructure with several machines to share the work load. In a real life scenario the scalability of the system assumes a much more impactful role considering the vast number of existing vehicles, and the number of certificate requests done by each vehicle in order to participate in V2X communications. We think that our contribution with the introduction of the RA Service is able to improve the interaction of the vehicles with the PKI in order to request such certificates and also increase the performance of the vehicle's authorization process by aggregating the requests and only validation the enrolment on the first authorization.

5.2 Future Work

During the development of this thesis some of the initially panned features could not be implemented due to the lack of time. In this section we review the main improvements to our V2X system that can be considered for future work. We start by the PKI Manager then move to the Vehicle Manager.

As explained in Chapter 1, one of out first objectives with this work was to extend mPKI, Multicert's production PKI, to issue the vehicular and CA certificates needed for V2X. This feature would be achieved through the usage of our V2X Library. However, due to the lack of time we decided not to do so and instead develop a demo PKI, the PKI Manager. Integrating with mPKI would give our system a more realistic approach in the interaction between vehicles and the PKI, allowing us to better understand the vehicular certificate issuing through our RA Service in a real case scenario.

Additional to the PKI Manager derive from this main limitation. The PKI Manager was developed with the purpose of demonstration, as such it only supports the most basic PKI operations of issuing certificates for the CAs and vehicles. The future work considered for this component is to add support for the renovation of CA certificates and the revocation of CA certificates and vehicle enrollment credentials through a CRL or OCSP mechanism. Such features are essential in a system that depends on identity management. Therefore, their implementation would approximate our PKI Manager to a production PKI in terms of functionality. We can also consider adding support for deleting certificates, keys and CAs through the backoffice platform. This feature would improve the testability of the system since it would be possible to create and delete a PKI just using the backoffic application.

The auditing of the RA service can be improved in the future by implementing application logs and introducing non-repudiation as a security feature by storing the vehicle's signed certificates requests in the database. Such features are of great importance in a real system because they allow the detection of security breaches, assessment of damages and aid in the recovery process.

Regarding the Vehicle Manager, the future improvements revolve around making the application more useful in terms of understanding the overheads of V2X in a more realistic way. To achieve this, the execution of the V2X communications within the Vehicle Manager would receive as input the output of the

network and traffic simulators presented in Section 2.4. Such improvements would allow us to estimate the network latency associated to the communication as well as the effect of cooperative messages in traffic management and safety.

Bibliography

[1] European commission mobility and transport. `https://ec.europa.eu/transport/road_safety/specialist/statistics_en`, 2016.

[2] Y. Lin, P. Wang, and M. Ma. Intelligent transportation system (its): Concept, challenge and opportunity. In *Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2017 IEEE 3rd International Conference on*, pages 167–172. IEEE, 2017.

[3] S. Rehman, M. A. Khan, T. Zia, and L. Zheng. Vehicular ad-hoc networks (vanets)—an overview and challenges. 3:29–38, 01 2013.

[4] E. C. Eze, S. Zhang, and E. Liu. Vehicular ad hoc networks (vanets): Current state, challenges, potentials and way forward. In *Automation and Computing (ICAC), 2014 20th International Conference on*, pages 176–181. IEEE, 2014.

[5] Ieee standard for wireless access in vehicular environments; security services for applications and management messages. `https://standards.ieee.org/findstds/standard/1609.2-2016.html`, 2016.

[6] Intelligent transport systems (its); security; security header and certificate formats. `http://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.02.01_60/ts_103097v010201p.pdf`, 2015.

[7] Volkswagen group assumes pioneering role in rapid road safety improvement. `https://www.volkswagen-newsroom.com/en/press-releases/volkswagen-group-assumes-pioneering-role-in-rapid-road-safety-improvement-541`.

[8] An assessment of lte-v2x (pc5) and 802.11p direct communications technologies for improved road safety in the eu. `https://5gaa.org/wp-content/uploads/2017/12/5GAA-Road-safety-FINAL2017-12-05.pdf`, .

[9] White paper on its spectrum utilization in the asia pacific region. `https://5gaa.org/wp-content/uploads/2018/07/5GAA_WhitePaper_ITS-spectrum-utilization-in-the-Asia-Pacific-Region_FINAL_160718docx.pdf`, .

[10] White paper on its spectrum utilization in the asia pacific region. `C-V2Xcontributestosaferroadsforeveryone:5GAAlivedemoeventinBerlin/`, .

[11] The v2x (vehicle-to-everything) communications ecosystem: 2019 − 2030 − opportunities, challenges, strategies forecasts. `http://www.snstelecom.com/v2x/`.

[12] N. Bißmeyer, H. Stübing, E. Schoch, S. Götz, J. P. Stotz, and B. Lonc. A generic public key infrastructure for securing car-to-x communication. In *18th ITS World Congress, Orlando, USA*, volume 14, 2011.

[13] B. Lonc and P. Cincilla. Cooperative its security framework: Standards and implementations progress in europe. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A*, pages 1–6. IEEE, 2016.

[14] W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn. A security credential management system for v2v communications. In *Vehicular Networking Conference (VNC), 2013 IEEE*, pages 1–8. IEEE, 2013.

[15] Intelligent transport systems (its); security; trust and privacy management. `http://www.etsi.org/deliver/etsi_ts/102900_102999/102941/01.01.01_60/ts_102941v010101p.pdf`, 2012.

[16] U.s. department of transportation. safety pilot model deployment. `http://safetypilot.umtri.umich.edu/`.

[17] Scms cv pilots documentation. `https://wiki.campllc.org/display/SCP/SCMS+CV+Pilots+Documentation`.

[18] Etsi automotive intelligent transport systems. `http://www.etsi.org/technologies-clusters/technologies/automotive-intelligent-transport`, 2017.

[19] S. K. Bhoi and P. M. Khilar. Vehicular communication: a survey. *IET Networks*, 3(3):204–217, 2013.

[20] J. E. Siegel, D. C. Erb, and S. E. Sarma. A survey of the connected vehicle landscape–architectures, enabling technologies, applications, and development areas. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[21] C. Sommer, J. Härri, F. Hrizi, B. Schünemann, and F. Dressler. Simulation tools and techniques for vehicular communications and applications. In *Vehicular ad hoc Networks*, pages 365–392. Springer, 2015.

[22] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[23] N. E. Lownes and R. B. Machemehl. Vissim: a multi-parameter sensitivity analysis. In *Proceedings of the 38th conference on Winter simulation*, pages 1406–1413. Winter Simulation Conference, 2006.

[24] J. L. Font, P. Iñigo, M. Domínguez, J. L. Sevillano, and C. Amaya. Architecture, design and source code comparison of ns-2 and ns-3 network simulators. In *Proceedings of the 2010 Spring Simulation Multiconference*, page 109. Society for Computer Simulation International, 2010.

[25] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[26] R. Barr, Z. J. Haas, and R. van Renesse. Jist: An efficient approach to simulation using virtual machines. *Software: Practice and Experience*, 35(6):539–576, 2005.

[27] C. Sommer, R. German, and F. Dressler. Bidirectionally coupled network and road traffic simulation for improved ivc analysis. *IEEE Transactions on Mobile Computing*, 10(1):3–15, 2011.

[28] itetris homepage. `http://www.ict-itetris.eu/`.

[29] B. Schünemann. V2x simulation runtime infrastructure vsimrti: An assessment tool to design smart traffic management systems. *Computer Networks*, 55(14):3189–3198, 2011.

[30] Java implementation of its intelligent transport systems (its) security header and certificate formats. `https://github.com/pvendil/c2c-commonl`, 2016.

[31] I. Recommendation. Itu-tx. 696. *Interfaces*, 10(20-X):49.

[32] Intelligent transport systems (its); application object identifier (its-aid); registration. `https://www.etsi.org/deliver/etsi_ts/102900_102999/102965/01.04.01_60/ts_102965v010401p.pdf`, 2018.

[33] Apache jmeter web page. `https://jmeter.apache.org/`.

[34] Apache jmeter documentation. `https://jmeter.apache.org/`.

[35] Instrumentation java documentation. `https://docs.oracle.com/javase/7/docs/api/java/lang/instrument/Instrumentation.html`.