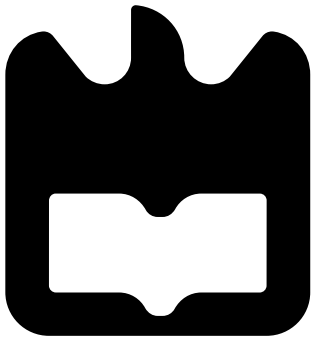
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2013

Daniel Belém de
Almeida Duarte

Implementação de Serviços de Segurança para Comunicações Veiculares

Implementation of Security Services for Vehicular Communications



Daniel Belém de Almeida Duarte

Implementação de Serviços de Segurança para Comunicações Veiculares

Implementation of Security Services for Vehicular Communications

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Joaquim Castro Ferreira, Professor Adjunto na Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Alexandre Manuel Moutela Nunes da Mota
Professor Associado, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Luis Miguel Pinho de Almeida
Professor Associado do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

Professor Doutor Joaquim Castro Ferreira
Professor Adjunto na Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro (co-orientador)

agradecimentos / acknowledgements

É com muito gosto que agradeço aos meus orientadores, Arnaldo Oliveira e Joaquim Ferreira, pela proposta de dissertação e ajuda ao longo deste trabalho.

Um especial agradecimento à minha família em particular aos meus pais e ao meu irmão por todo o apoio e motivação ao longo da minha formação académica.

Agradeço também à Ana por me apoiar nos meus piores momentos e por me ter dado força durante esta caminhada.

Quero também agradecer a todos os meus amigos que me proporcionaram bons momentos ao longo do meu percurso académico. Às pessoas do laboratório de Radio Frequência pelo bom ambiente proporcionado ao longo do meu trabalho.

Palavras-chave

Segurança, Algoritmos de Segurança, ECDSA, IEEE 1609.2, Comunicações Veiculares.

Resumo

Ao longo dos últimos anos tem existido uma enorme evolução nas redes veiculares com o objectivo de desenvolver protocolos e protótipos que satisfaçam os requisitos do sistema de transporte inteligente. Uma das normas desenvolvidas é a norma IEEE 802.11p que define a camada física para a criação de uma plataforma que permita a formação de uma rede veicular. A VANET (Vehicular Ad-Hoc Network) é uma rede criada pelos vários elementos da estrada, como carros e plataformas de prestação de serviços encontradas ao longo da estrada. Estas redes são de elevada importância, permitindo fornecer vários tipos de serviços: proporcionar maior segurança aos condutores e ocupantes reduzindo o número de acidentes; aumentar a eficiência rodoviária reduzindo o impacto no ambiente; proporcionar serviços de navegação e entretenimento para os seus ocupantes.

O potencial destas redes é enorme, mas exige requisitos de segurança e anonimato de forma a não serem adulteradas por atacantes que pretendam tirar partido destas. Deste modo, estudos mais recentes têm dado um maior relevo às camadas superiores da pilha protocolar, nomeadamente ao estudo do impacto da segurança nestas redes.

Neste trabalho propõe-se criar uma arquitetura e a analisar o impacto que os serviços de segurança têm quando adicionados a uma rede veicular. Deste modo o Standard IEEE 1609.2 aparece como forma de colmatar as falhas de segurança que possam existir numa rede veicular.

Foi efetuada a implementação de serviços de segurança com base no Standard IEEE 1609.2 D17 Draft Standard implementando os vários algoritmos criptográficos específicos assim como os protocolos para gerir a troca de mensagens seguras na rede. Esta implementação foi desenvolvida em software com a ajuda da livraria OpenSSL para implementação dos algoritmos de segurança. O algoritmo criptográfico ECDSA que garante autenticação de mensagens é mantatório para todas as mensagens trocadas tendo sido este o foco da implementação.

De modo a ser possível testar a implementação num ambiente real, foi efetuada a integração do sistema com outros módulos tais como a geração de mensagens e o protocolo de transporte WSMP.

O objetivo desta dissertação é avaliar o desempenho do sistema e o "overhead" causado na rede quando os serviços de segurança são adicionados a uma rede veicular. Como On Boad Unit (OBU) foi considerado o uso de um Raspberry-Pi, concluindo que uma implementação puramente em software não é viável conseguindo apenas atingir o número máximo de 40 verificações de assinaturas por segundo usando o algoritmo criptográfico ECDSA.

Keywords Security, Cryptography Algorithms, ECDSA, IEEE 1609.2, Vehicular Communications.

Abstract

Over the last few years there has been a considerable development in the field of vehicular communications (VC) in order to develop standards and prototypes that satisfy the requirements of the Intelligent Transportation System (ITS). One of these standards is the IEEE 802.11p that defines the physical layers to create a platform for vehicular communications.

Cars and elements on the road are viewed as wireless routers (nodes), creating a VANET (Vehicular Ad-Hoc Network), which is part of the ITS. It's intention is to provide several services as: a safer environment for drivers by reducing the number of accidents/injuries; improve traffic congestion and consequently reduce the impact of cars in the environment; provide infotainment services.

These networks have promising features to guarantee a high level of safety between drivers, however they have to work on a secure and anonymous way since they can be threatened and attacked by malicious sources. Therefore more recently studies have focused on studying the impact of security on VC.

In this dissertation it is proposed the creation of an architecture and the analysis of the impact of security services when they are added to VANET. This is achieved by using the IEEE 1609.2 Standard to overcome the limitations of security on vehicular communications.

An implementation of the required cryptographic algorithms and protocols to manage the sharing of secure messages according to the IEEE 1609.2 Standard was developed with the help of the OpenSSL library. The ECDSA cryptographic algorithm ensures the authentication of all messages, which is the focus of this dissertation.

In order to achieve an architecture capable of being integrated and tested in a real scenario, the implemented system was joined with other applications as the WSMP (WAVE Short Message Protocol) and the generation of CAM messages. With this integration it was possible to evaluate the overhead that is caused when the process to sign/verify a digital message is added to a vehicular communication. For these tests a Raspberry-Pi was used as a On Board Unit, concluding that a pure software implementation is not feasible, allowing only a maximum number of 40 signature verifications/second using the ECDSA cryptographic algorithm.

Contents

List of Figures

List of Tables

Acronyms

AES Advanced Encryption Standard

API Application Programming Interface

BSM Basic Safety Message

CA Certificate Authority

CAM Cooperative Awareness Message

CCH Control Channel

DENM Decentralized Environmental Notification Message

DES Data Encryption Standard

DH Diffie–Hellman

DL Discrete Logarithm

DoS Denial of Service

DSA Digital Signature Algorithm

DSL Digital Subscriber Line

DSRC Dedicated Short Range Communications

DSS Digital Signature Standard

ECC Elliptic Curve Cryptography

ECDSA Elliptic Curve Digital Signature Algorithm

ECU Electronic Control Unit

ETSI European Telecommunications Standards Institute

FAST FIX Adapted for STreaming

FCC Federal Communications Commission

FIPS Federal Information Processing Standard

FPGA Field-Programmable Gate Array

GPS Global Positioning System

HSM Hardware Security Model

I2V Infrastructure to Vehicle

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

IPv6 Internet Protocol version 6

ITS Intelligent Transportation System

MAC Medium Access Control

NIST National Institute of Standards and Technology

OBD On Board Diagnostics

OBU On Board Unit

OFDM Orthogonal Frequency-Division Multiplexing

OSI Open Systems Interconnection

PHY Physical Layer

PKI Public-Key Infrastructure

PSID Provider Service IDentifier

RF Radio Frequency

RSI Road Side Infrastructure

RSU Road Side Unit

SAE Society of Automotive Engineers

SCH Service Channel

SeVeCom Secure Vehicular Communication

SHA Secure Hash Algorithm

USB Universal Serial Bus

V2I Vehicle to Infrastructure

V2R Vehicle to Road Side

V2V Vehicle to Vehicle

VANET Vehicular Ad-Hoc Network

VC Vehicular Communications

WAV Waveform Audio File Format

WAVE Wireless Access in Vehicular Environments

WME Wave Management Entity

WSA WAVE Service Advertisement

WSM WAVE Short Message

WSMP WAVE Short Message Protocol

Chapter 1

Introduction

1.1 Motivation

Nowadays the number of cars and people travelling is increasing and the safety among travellers is a constant concern for everyone. According to [1] in 2010 the number of vehicles in the world has reached the number of 1.015 billion, with an approximate ratio of 1:7 cars per person.

In the USA the number of accidents per year is about 5.25 million, causing many deaths and injuries on the drivers. This number is a big concern for the authorities and it is expected to rise in the next years so measures must be taken in order to reduce and improve the safety of drivers.

The Federal Communications Commission (FCC) [2] in the USA allocated a 75 MHz of spectrum in the 5.9GHz band to be used in Intelligent Transportation System (ITS). This is also referred as the 5.9-Dedicated Short Range Communications (DSRC) and its main goal is to provide vehicle to vehicle (V2V) and vehicle to infrastructure (V2I) communication. So, services and applications should be developed to help prevent accidents and manage traffic flow. A fast exchange of a special type of messages is shared among vehicles and the infrastructure to provide safety information to the participants in the communication. Therefore a group of standards was defined both in the USA and in Europe to define the DSRC for vehicular communications, named Wireless Access in Vehicular Environments (WAVE) and ETSI-ITS respectively.

The IEEE 1609 Working Group defines a set of standards for Wireless Access in Vehicular Environments (WAVE). This family of standards define an architecture, a set of interfaces and services that all together enable secure Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) wireless communications. The IEEE 1609.2 is the standard addressing security services that will prevent attacks and will ensure that the communications are made securely between nodes of the network.

Vehicular Ad-Hoc Network (VANET) are very sensitive to security threats because the more popular a system becomes, the higher the number of attacks and vulnerabilities. These networks will be analysed by malicious people whose intent is to take advantage: from causing the chaos on the road, by intentionally redirecting traffic flow, to false drivers alerts that may cause accidents.

Therefore, the objective of this dissertation is to analyse the latest version available at the time, the IEEE 1609.2 D17 Draft Standard and to develop an architecture that allows a secure communication on a vehicular environment.

1.2 Objectives

Security is a major problem in every common and widely used system and VANET are no exception. For the specific case of vehicular communications, the IEEE 1609.2 standard comes to define the cryptography algorithms and services that should be used in order to ensure a secure environment. Also the IEEE 1609.3 standard defining the communication protocols to use in Vehicular Communications (VC) have an important role in monitoring traffic patterns and responding to possible attacks. A secure system depends therefore on much more than just cryptography algorithms, so a group of defined protocols and a well structured design are also needed [3].

When well defined architecture and secure protocols are used together with the correct cryptography algorithms, it should allow authentication, anonymity, confidentiality, integrity, non-repudiation and access control.

For this dissertation the following objectives were defined:

- To study the cryptography algorithms used in vehicular communications;
- To study the requirements in the IEEE 1609.2 D17 draft standard;
- To implement a software version of the secure protocols;
- To assess the system performance;
- To integrate WAVE Short Message Protocol (WSMP), CAM message generation and security;

1.3 Structure

This dissertation is organized as follows:

Chapter 2 - Background
This chapter starts with an introduction of the ITS by describing the standards and the architecture model of a vehicular communication system. Then, detailed information is given on how security is implemented in VANET. Finally the IEEE 1609.2 D17 Draft Standard is analysed to better understand the architecture and the requirements to propose an implementation.

Chapter 3 - IEEE 1609.2 Implementation
In this chapter the proposed implementation of the IEEE 1609.2 D17 Draft Standard is explained in detail. The security model is presented: explaining how the cryptographic algorithms were implemented and how the secure protocols to correctly sign a message are

applied and implemented.

Chapter 4 - Integration with WSMP and Facilities Layer

In this chapter the Security Services are inserted into a "black-box" that contains other modules as the WSMP and the facilities application (generation of Cooperative Awareness Message (CAM) message). The implementation of this black box was done in order to achieve a software that works as desired. A description of how this integration was done is also shown.

Chapter 5 - Experimental results

In this chapter a discuss of the results obtained from the implementation of the security services is presented.

Laboratory experiments of the integration with the WSMP and the facilities application are also performed to benchmark the whole system.

Chapter 6 - Conclusions and Future Work

A summary of the dissertation is done and future work is proposed.

Chapter 2

Background

2.1 Introduction

This dissertation is part of two research projects HEADWAY-Highway Environment ADvanced WArning sYstem, funded by Brisa, a motorway operator, and ICSI - Intelligent Cooperative Sensing for Improved traffic efficiency, an FP7 project. These projects have been developed in the Telecommunications Institute in Aveiro and its main goal is to develop a FPGA-based softcore implementation of a IEEE 802.11 A6 / ETSI ITS G5 controller, the IT^{2}S platform. This work is integrated in these projects but focusing on an architecture capable of securing the communications on Vehicular Communications (VC) instead of working on the lower layers.

Therefore this chapter gives an introduction to the Intelligent Transportation System (ITS) explaining all the required standards and how security is addressed. The specific case of security in VC is analysed detailing its infrastructure, challenges, types of attacks and requirements. Theory behind the necessary cryptography algorithms in VC is also presented to better understand its context. Also some related work and projects addressing security on VC are referred.

2.2 Intelligent Transportation System

Vehicular networks, most of the times named as Vehicular Ad-Hoc Network (VANET), are a part of the ITS that use moving cars and the infrastructure as nodes to create a network. Cars are viewed as a wireless router allowing all cars within a radius of approximately 1Km to communicate with each other. The aim of the ITS is to provide useful information to its users by creating a safer environment and by improving traffic flow. To support these services the VANET networks were designed to work on vehicular environments allowing communications between cars even when they are travelling at high speeds.

According to [4] "In the following years it is expected a significant increase in vehicular Dedicated Short Range Communications (DSRC) supporting safety, comfort and infotainment services. However, to be widely adopted and used, vehicular DSRC protocols and technologies must allow Vehicle to Vehicle (V2V), Vehicle to Infrastructure (V2I) and Infrastructure to

Vehicle (I2V) communications in very heterogeneous vehicular scenarios and real-time features to effectively support safety critical services".

2.2.1 Network Infrastructure and Nodes

The network infrastructure of a vehicular network is composed of nodes, which can be an On Board Unit (OBU) or a Road Side Unit (RSU). These nodes are placed either in each car(OBU) or on the side of the road (RSU). The communication can be done between OBU to Road Side Infrastructure (RSI) or from OBU to another OBU, which represents a V2I or V2V communication respectively. RSU will have a fixed place on the road and its main goal is to retrieve or spread important information to the OBU nodes as, for example, information about the road, the weather or the traffic. A backbone connection is associated to the RSU to get access to the Internet through mobile communication, Digital Subscriber Line (DSL) or fibre [5].

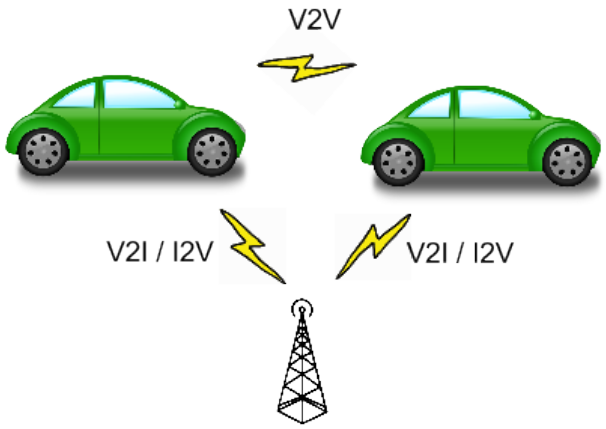


Figure 2.1: Vehicle Communication Types.

The purpose of VC is to provide safety information by sharing important messages among cars, allowing the following services:

- Traffic information;
- Traffic management and warnings;
- Reduce the impact of cars in the environment;
- Global Positioning System (GPS) based services;
- Electronic payment;

A network infrastructure like VANET can have several hundred million of nodes all over the world making it one of the biggest ones.

2.2.2 DSRC - Dedicated Short-Range Communication

This technology has been developed since 1995 and the first DSRC standard was released in 1998 and it is already standardized in the IEEE 802.11p. Its aim is to provide safe and private communication services between cars and road side units [6].

The DSRC characteristics are[5]:

- Standardized on IEEE 802.11p;
- Address the physical(PHY) and Medium Access Control (MAC) layers;
- 7 licenses channels in 5.9GHz, 10MHz each;
- Data rates from 6 up to 27 Mbps;
- Communications up to 1000m;
- Security using Public Key infrastructure;
- Low latency and time-critical responses ($\approx$ 50ms);
- Technology regulated by the Federal Communications Commission (FCC) and IEEE Standards;

In USA the DSRC has a frequency range from 5.850 to 5.925 GHz, divided in 7 channels (10MHz each) starting on channel 172 and ending on channel 184 with a 5Mhz reserved as guard band. From these 7 channels, one channel is configured as Control Channel (CCH) and all the other 6 are configured as Service Channel (SCH). High priority messages are carried by the control channel and all the others are sent through the service channels.

In Europe the allocation is different, having a range of 30 MHz allocated between the 5.875 GHz to 5.90 GHz.

In figure 2.2 the DSRC allocation in different world areas is presented.

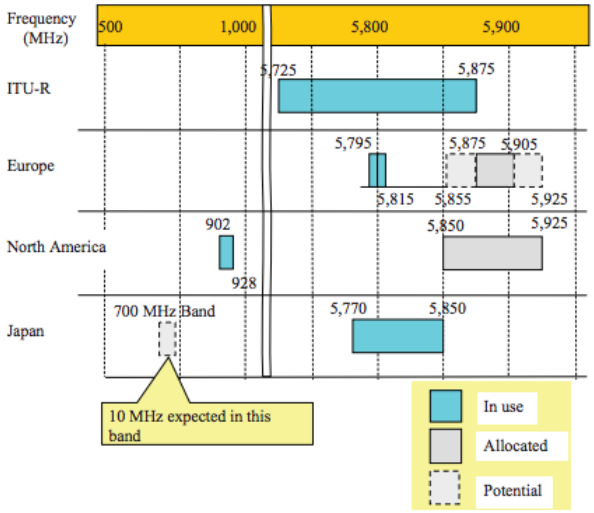


Figure 2.2: DSRC worldwide spectrum allocation [7]

2.3 Vehicular Communication Standards

In order to have a vehicular communication system, a set of standards needed to be defined but different approaches have been taken around the world. Most of the approaches are based on the American Standard IEEE 802.11p (recently named as 802.11:2012 A6) addressing the lower layers as the PHY and the MAC Layers of the stack. The main differences in the approaches are noticed when the higher layers are referred: differences in the application layer, the shared messages type and the communication protocol.

2.3.1 IEEE WAVE Protocol Stack

The Wireless Access in Vehicular Environments (WAVE) is the American vehicle based communication group of standards and its protocol stack is composed of multiple parts, from several Institute of Electrical and Electronics Engineers (IEEE) standard families. The IEEE 802.11p, the IEEE 1609.x family and the Society of Automotive Engineers (SAE) define what is called the WAVE protocol stack.

In figure 2.3 the WAVE stack is presented. This stack has some key components that are necessary to allow the WAVE to work correctly: multi-channel operations, networking and security services and resource manager which are described ahead.

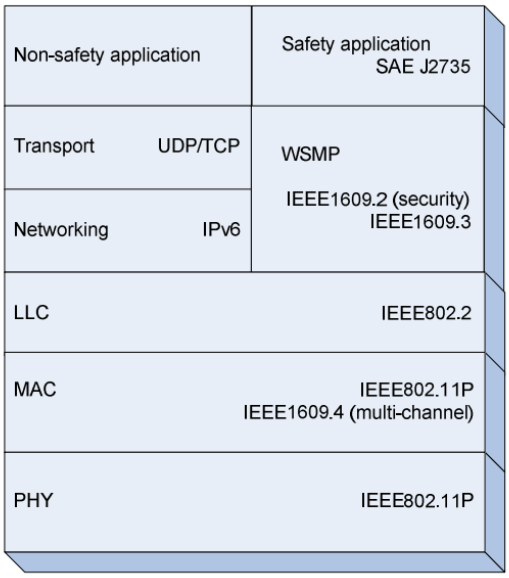


Figure 2.3: WAVE Protocol Stack [8].

IEEE 802.11p

The IEEE 802.11p is an amendment to the well known wireless standard 802.11a/b/g/n (Wi-Fi). However the modulation adopted is Orthogonal Frequency-Division Multiplexing (OFDM) on 10-Mhz channels in the 5.9 GHz band which is different from the normal 20-MHz channels used on the Wi-Fi [2] [8]. The reason for this amendment is because VC have

special needs, requiring low latencies, reduced inter-symbol interference due to the multi-path propagation and the Doppler shift effect.

In figure 2.4 the differences between Wi-Fi and WAVE are presented.

Parameters	WAVE	Wi-Fi
Frequency Band	5.9 GHz	5/2.4 GHz
Channel Bandwidth	10 MHz	20 MHz
Supported Data Rate (Mbps)	3, 4.5, 6, 9, 12, 18, 24, and 27	6, 9, 12, 18, 24, 36, 48 and 54
Modulation	Same as Wi-Fi	BPSK, QPSK, 16QAM and 64QAM
Channel Coding	Same as Wi-Fi	Convolutional coding rate: 1/2, 2/3 and 3/4
No. of Data Subcarriers	Same as Wi-Fi	48
No. of Pilot Subcarriers	Same as Wi-Fi	4
No. of Virtual Subcarriers	Same as Wi-Fi	12
FFT/IFFT Size	Same as Wi-Fi	64
FFT/IFFT Interval	6.4 µS	3.2 µS
Subcarrier Spacing	0.15625 MHz	0.3125 MHz
CP Interval	1.6 µS	0.8 µS
OFDM Symbol Interval	8 µS	4 µS

Figure 2.4: Wi-Fi and WAVE Characteristics [8].

IEEE 1609.1

This standard defines the interfaces, services and packets for the resource manager in WAVE.

IEEE 1609.2

This is the most advanced automotive standard related with security. Its aim is to secure all the communications that are performed in a vehicular communication. The standard defines which security mechanisms should be used and how the packets should be correctly formatted.

A set of protocols are also needed to correctly process the messages because security is not only about cryptography mechanisms [3]. A special type of certificates called WAVE-Certificates are also referred in the standard as a compact certificate type special for vehicular communications.

IEEE 1609.3

The networking services are defined by this standard as an additional communication protocol to the Internet Protocol version 6 (IPv6), a non-ip protocol specific for WAVE communications. It is called as the WAVE Short Message Protocol (WSMP), which is able to operate over the Service and Communication channels (SCH,CCH).

Vehicular communications main requirements are low latency on communications as cars are travelling very fast. So, the purpose of developing such communication protocol was to

take down some of the overhead that was imposed by an internet protocol.

The IPv6 packet takes an overhead of 40 bytes which is a large packet compared with only 11 bytes that the WSMP carries [9] [10].

In the figures 2.5 and 2.6 the size of each field and the differences of both packet headers are shown.

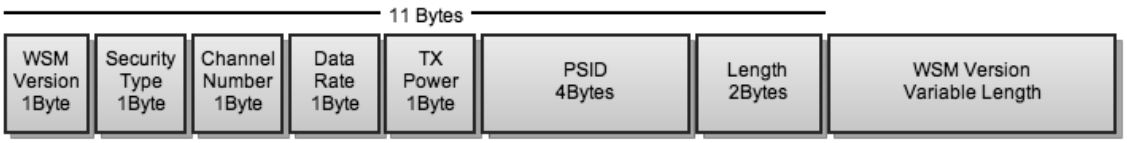


Figure 2.5: WSMP Packet Structure.

- **WAVE Short Message (WSM) version** - Defines the version of the implemented WSMP standard.
- **Security Type** - Identifies if the packet is unsecured, signed or encrypted.
- **Channel Number**, **Data Rate**, **TX Power** - Allow control of the radio parameters.
- **Provider Service IDentifier (PSID)** - Works as and UDP and TCP port to identify the services that are being transmitted.
- **Length** - Indicates the number of bytes that the WSM data carries.
- **WSM Data** - The Message to be transmitted.

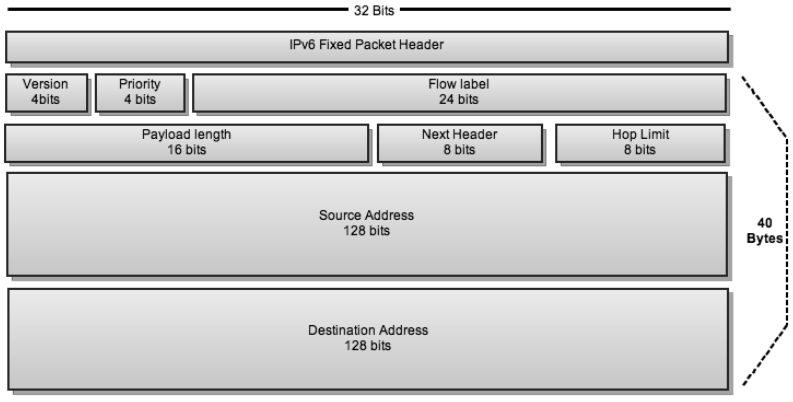


Figure 2.6: IPv6 Packet Header.

IEEE 1609.4

This standard provides frequency band coordination and management as e.g. the multi-channel operation in the DSRC.

SAE J2735

This standard is defined by the SAE, which defines parts of the higher layers of the WAVE Open Systems Interconnection (OSI) model. It defines the message structure of the messages shared among elements on the road whose purpose is to share important information about the traffic or the state of the vehicles.

According to Hartenstein *et all* [5] one of the most important message is the Basic Safety Message (BSM), which aim is to provide vital information about the vehicle: velocity, position, direction, etc. These messages are shared among all vehicles on the road and it's the way to get information about the position of every car. This information can therefore be used to prevent, for example, accidents by providing vital information to the driver.

2.3.2 ETSI Standards

The European Telecommunications Standards Institute (ETSI) defines a European stack for VC in a similar way to the WAVE Standards in the USA. Some of the standards defined by the ETSI can be similar to the USA standards, while others have big differences.

In figure 2.7 the layers which compose the ETSI stack are presented with a small description of each.

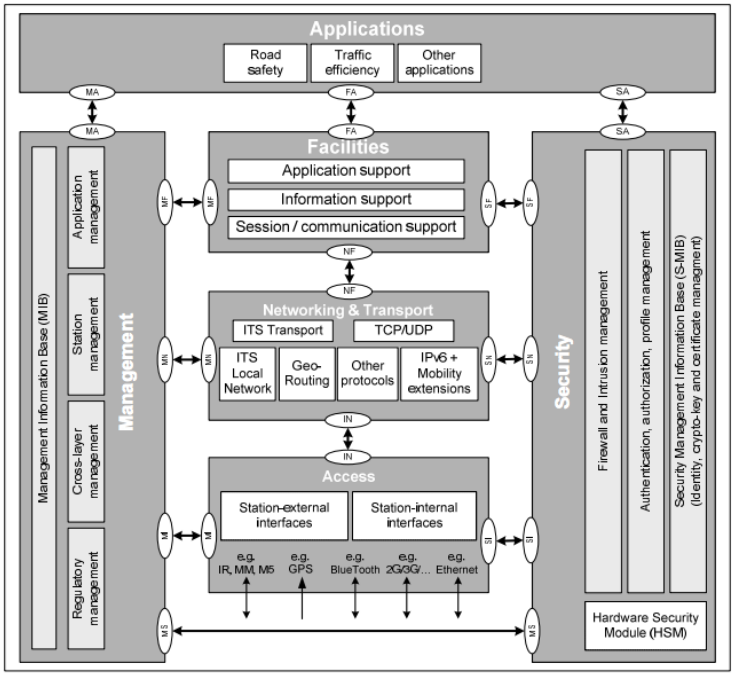


Figure 2.7: ETSI ITS Stack [11]

Access Layer
This layer defines the PHY and MAC layers along with the interfaces to other layers on the stack.

Networking & Transport Layer
This layer is responsible for the management of the network and the data flow. This layer is capable of supporting multiple network protocols as GeoNetworking, CALM FAST, IPv6 or other protocols.

Facilities layer
The facilities layer is responsible for the data retrieval from the vehicle or from the network and provides it to the application layer. It is also responsible for the acquisition of data collected from the car sensors and generates adequate messages with the retrieved information.

Security
The security layer is responsible for providing adequate security to all the layers in the stack. As the ETSI Security Standard is not still well defined and documented it was decided to implement and base this study on the American IEEE 1609.2 D17 Draft Standard.

Management
This layer is responsible for the management of all the layers in the ETSI stack.

2.4 Cryptography Algorithms

Cryptography is part of our daily life (using the Internet, playing a DVD, etc) and we barely don't notice it. With the increase of technology and the constant connection to the internet, the privacy and security worries everyone and measures have to be taken to protect personal information.

The first use of cryptography was made by the ancient Greeks who used a *scytale transposition cipher* - a thin cylinder made out of wood (figure 2.8). The Spartan army used this cryptography method to send and receive sensitive messages. The way this worked was by wrapping around the stick with a paper-strip and write the message, this way when the paper-strip was removed, the message seamed to be only scrambled letters [12].

Figure 2.8: Scytale Transposition Cipher [13]

Nowadays these algorithms are no longer used because computers can easily crack them. Computer based cryptography is now used to allow present and future security.

Computer cryptography schemes generally belong to one of the following categories:

- Symmetric key algorithms
- Public Key algorithms

2.4.1 Symmetric Key Algorithms

In this type of algorithms, both sides need to share de same key, otherwise the communication cannot be performed. This means that a previous knowledge about the participants (computers) in the communication must be known. A secret key is given to each of the participants so that they are able to encrypt and decrypt messages.

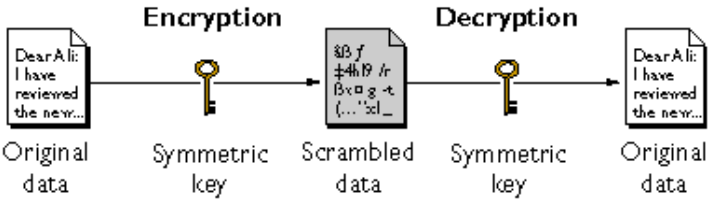


Figure 2.9: Symmetric Security Scheme - Encryption of Data [14]

The first symmetric encryption algorithm was Data Encryption Standard (DES) which used a 56-bit key [15]. As computers have become faster, nowadays this standard is no longer used and the adopted standard is now Advanced Encryption Standard (AES), using 128, 192 or 256-bit keys, which is estimated to fully satisfy the security requirements for the upcoming years [16].

2.4.2 Public Key Algorithms

The Public Key algorithms, also named as asymmetric algorithms, make use of a combination of two keys. A private key, kept in secret for the user, and a public key to be spread for all the users in the communication. The private and public key (key pair) are related but in a way that given the public key it is computationally infeasible to compute the private key. To better understand the concept of this algorithms let us consider the following example:

Consider two entities, Alice and Bob who are going to communicate. Both of them generate a key pair, private and public key and exchange the public key. From this point of the communication, if Alice wants to send a secret message to Bob, she will encrypt the message with Bob's public key and if Bob wants to communicate with Alice he will use Alice's public key to cipher the message. This way the recipient of the ciphered data will be able to decrypt data using its own private key [17].

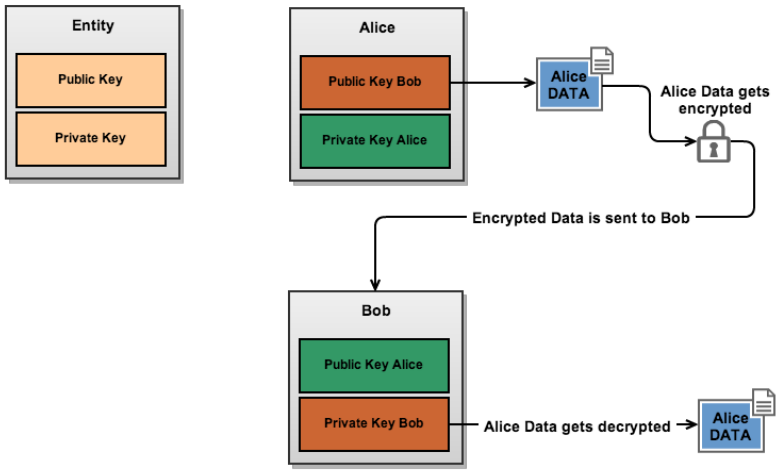


Figure 2.10: Public-Key Scheme for Encryption.

Public Key algorithms may be used to encrypt messages (e.g.RSA algorithm) or to sign a message in order to provide authenticity (Digital Signature Algorithms).
The use of Digital Signatures is a way to guarantee that a certain message came from the expected person. In this case the public key is shared with everyone and a message is signed with its corresponding private key. This way everyone who has the public key is able to verify the message, proving that the sender had access to the private key and is probably the expected person.

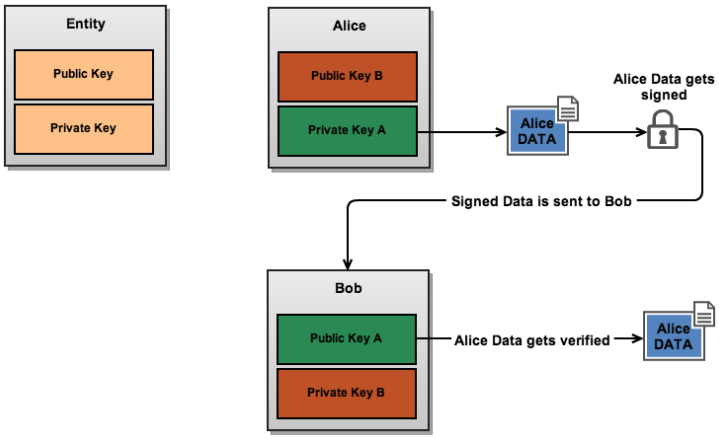


Figure 2.11: Public-Key Scheme for Digital Signatures.

In figure 2.10 and 2.11 both public key schemes for encryption and for digital signature are described.

The security on public key algorithms is based on some computation problems that are listed bellow [5]:

- The integer factorization problem, when given a positive integer it is computationally hard to find its prime factorization.
- The discrete logarithm problem, when given g and h it is hard to find x which is the solution for the following equation: $g^x = h$.
- Based on elliptic curves algebraic problems.

These mathematical problems are used in the generation of private and public key, guaranteeing that it is computationally infeasible to derive the private key from the public key.

The main differences in these 3 types of computational problems is that each of them needs a specific key size that ensures the infeasibility of the problem to derive one key from the other. For example, a RSA algorithm that uses the *integer factorization problem* requires a 1024-bit key length size to provide the same level of security as an Elliptic Curve Cryptography (ECC) with 160-bit key length. In table 2.1 a comparison of key sizes to provide the same level of security between RSA and ECC is presented.

Key Strength Comparisons

RSA(bits)	ECC(bits)
1024	160
2048	282
4096	409

Table 2.1: Key Strength Comparisons [18].

RSA is more common than ECC and there are several implementations of it, although the length of the key size makes it unsuitable for the vehicular communication because of the overhead caused by it. This way the ECC is referred in the IEEE 1609.2 Standard as the choice for WAVE.

RSA

The RSA algorithm (named after its inventors Rivest, Shamir and Adleman) was proposed in 1978 and it is one of the most versatile public-key algorithms. It is suitable for encryption/decryption, signing/verification and for key establishment. Its security is based on the difficulty of factoring large integers, and the recommended RSA key length should be at least 1024 bits long to provide adequate security [19].

DH - Diffie–Hellman

The Diffie–Hellman (DH) algorithm is a specific method of exchanging cryptographic keys. Each of the peers uses its own private key and the public key of the other peer to generate a symmetric key that no third-party can use. The security of this algorithm is based on the complexity of computing logarithms in a finite field. The recommended length of the key to be used is at least 1024 bits long to provide adequate security. [19]

DSA - Digital Signature Algorithm

The Digital Signature Algorithm (DSA) is a Discrete Logarithm (DL) Scheme and it was proposed by U.S. National Institute of Standards and Technology (NIST) and was specified by the U.S. Government Federal Information Processing Standard (FIPS 186) called the Digital Signature Standard (DSS) [20].

The DSA was exclusively designed for signing/verification and also for data integrity.

The security of this algorithm is based on the complexity of computing logarithms in a finite field. The recommended length of the key to be used is at least 1024 bits long to provide adequate security [19].

Key generation

In the DL systems the key pair(y, x), public key (y) and private key(x) are associated with 3 domain parameters(p, q, g). These domain parameters are a prime p, q is a prime divisor of $p - 1$ and $g \in [1, p - 1]$ has order q ($t = q$ is the smallest positive integer satisfying $g^t \equiv 1 \pmod{p}$).

A private and a public key are derived from this parameters, the private key(x) is an integer selected randomly that satisfies the equation $x \in R : [1, q - 1]$ and the public key (y) is obtained by $y = g^x \pmod{p}$.

The DL systems are very powerful because of the mathematical problem in determining the private key (x) given the domain parameter (p, q, g) and the public key (y) [17].

Algorithm 1 DL Domain Parameters

Require: Security Parameters l, t

1: Select a t-bit prime q and an l-bit prime p such that q divides p-1.
2: Select an element g of order q.
3: Select arbitrary $h \in$ [1, p - 1] and compute $g = h^{(p-1)/q} \bmod p$.
4: **if** $g = 1$ **then**
5: go to step 3.
6: **else**
7: Return **parameters** (p, q, g).
8: **end if**

Algorithm 2 DL Key Pair Generation

Require: Domain Parameters (p, q, g).

1: Select $x \in R[1, q-1]$

2: Compute $y = g^x \bmod p$.

3: Return **key pair** (y, x).

Signature

In the DSA an entity with the private key x signs a message and its identity can be verified by any other entity who has access to the public key y. The algorithm is described bellow where H is the Hash function used to generate the message digest of the input message.

Algorithm 3 DSA Signature Generation

Require: Domain Parameters (p, q, g), private key x, message m.

1: Select $k \in R[1, q-1]$

2: Compute $r = (g^k \bmod p) \bmod q$.

3: **if** $r = 0$ **then**

4: Go to step 2.

5: **else**

6: Compute $s = k^{-1} (H(m) + xr) \bmod q$.

7: **end if**

8: **if** $s = 0$ **then**

9: Go to step 2.

10: **else**

11: Return **Signature** (r, s).

12: **end if**

Algorithm 4 DSA Signature Verification

Require: Domain Parameters (p, q, g), public key y, message m, signature (r, s).

1: Verify if r, s are integers in the interval [1, q-1].
2: If any parameters verification fails the **Rejected Signature**
3: Compute $h = \mathrm{H}(m)$.
4: Compute $w = s^{-1} \bmod q$.
5: Compute $u_1 = hw \bmod q$.
6: Compute $u_2 = rw \bmod q$.
7: Compute $r' = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
8: **if** $r = r'$ **then**
9: Return **Valid Signature**.
10: **else**
11: Return **Rejected Signature**.
12: **end if**

2.4.3 HASH Algorithms

The HASH Algorithms have the ability to transform big amounts of data into a fixed bit array block. This type of functions can be seen as a fingerprint of a certain information. There are several HASH functions as SHA, MD5 or MD2, but the one required for this work is the Secure Hash Algorithm (SHA) [21].

SHA-1

The SHA-1 is the original version of the SHA algorithm. It was especially design to be used with the DSA, but it can be used with any other public-key algorithm. Its design is very similar to other hash functions as MD2 or MD5. This function is most of the times referred as SHA-1 or SHA-160, because the size of the *fingerprint* is 160-bits. The number of bits used in this functions is expected to provide adequate security for the following years[22].

SHA-256/SHA-224

The SHA-256 is very similar to the SHA-1 and the main difference is on the output size of the fingerprint. The name reference to 256 is the size of the output bit array which in this case is 256 bit long. The DSA algorithms in this standard required a 224 and 256 bit hash algorithms. The SHA-224 can be obtained the same way as SHA-256 but the final value is truncated to 224 bits [22].

2.4.4 Certificates and Authentication

A certificate is a way to identify an entity, to ensure that a certain public-key is indeed from the expected person. Certificates are used to fill the gap we had when sharing a public-key which everyone could use but nobody really knew who the person really sharing it was.

For example, if we want to get a driving license (certificate), some tests are made to make sure we are able to drive. The authority that verifies our identity and ensures our ability to drive is a Certificate Authority (CA). A CA is an entity which verifies identities and issues certificates. These certificates have a relation with the public key and the name of the entity who requests the certificate. Certificates help prevent attacks from people who create fake public key and try to identify themselves as the real sender, this way only the public key that was certified by the certificate will work with the corresponding private key [14].

A digital certificate is composed of 3 parts, the identification of the user, the associated public key and the signature.

A special certificate for vehicular communications is needed in order to have small certificates that can be sent over the network without causing a significant overhead in the network. In figure 2.12 the specific fields of how a WAVE-Certificate should look like and also the relation between a Root and a Sub-Ordinate Certificate is shown.

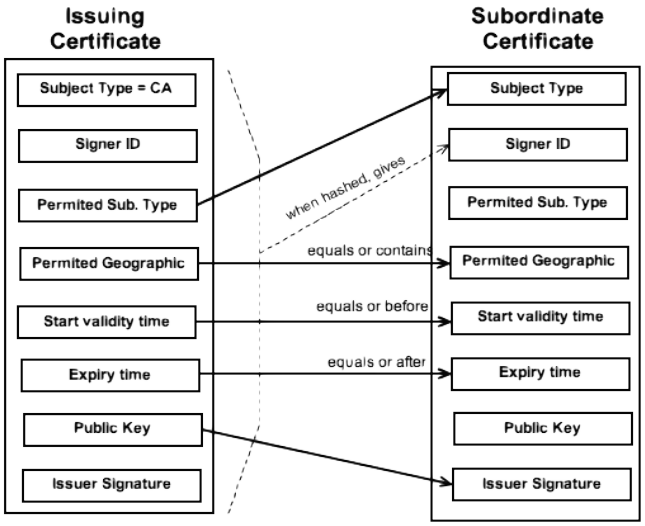


Figure 2.12: WAVE Certificates - Structure and relation between Root and Sub-Ordinate Certificates [21].

Authentication

Authentication is the core security requirement in a vehicular network, so it is required that all applications use authentication in their messages. It is also mandatory that each message carries a location and time stamp to avoid replay attacks. Authentication is therefore the way to confirm the identity of an entity by means of digital signatures. The Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm is the chosen algorithm by the IEEE 1609.2 Standard as the algorithm to provide digital sigantures on VC. In the the next section the

ECDSA algorithm is presented.

2.4.5 ECDSA - Elliptic Curve Digital Signature Algorithm

The ECDSA is used to create a digital signature in order to allow authenticity. A digital signature can be seen as a normal handwritten signature that we can recognize but we can not forge. The ECDSA is a digital signature algorithm that works over an Elliptic Curve. This elliptic curve is a function in which points on this curve will be used to create the signature and consequently verify it. This is all about mathematics, so private and public keys are points on this curve which when used on a certain equation can generate or verify a signature.

Without getting into deeper mathematics, basically two random points on the curve are generated, one is the considered "point of origin" and the other one is the private key. When the private key is used with the "point of origin" in a special equation it gives another point on the curve which is the public key. At this point we have our key pair generated and when we want to sign a message we use the private key and the hash of the data in a special equation that will generate the signature. This signature is composed of two parts, named by R and S. In order to verify a signature we use the public key and one part of the signature (S) with a special equation. In case of success it will result on the same value as R.

Summing up, we use a private key along with data to generate a signature (R and S) and if the mathematical equation used with the public key and S results in R, the signature is verified.

To illustrate how this algorithm works, two users are going to be considered, commonly named by Alice and Bob. Alice is going to communicate with Bob and so Alice generates a key-pair as in algorithm 2, which is a generation of a private and a public key. Alice then performs the signature generation process (algorithm 3) and then she publishes her public key. Then Bob, who is going to receive Alice message, can verify the signature on the message by first getting Alice's public key and then by performing the signature verification process (algorithm 4).

2.5 Security in Vehicular Communications

The automotive industry is in constant development and the main evolution that cars have suffered is related with information technology, in which safety applications will have an important role in the upcoming years. These applications will take an important role in creating a safer environment for all drivers, guaranteeing that potential threats that may harm the driver are detected in advance and some action may be taken to prevent it. Security comes on top of these services to guarantee that there are no malicious attacks nor any type of manipulation of these systems.

Security objectives and solutions are very well defined for computer based architecture in general but for vehicular environments the approach needs to be different [5]:

- The computational performance is quite small in the embedded computers that the vehicles will carry. These low-cost processors may not have enough memory and performance to allow the cryptographic operations that may grant security.

- A car has a life-time of at least 10 years, and the upgrade of the secure system or the OBU can not be granted to be upgraded in this time. It is important that the OBU works and fulfils all the security requirements for the car life-time.

- An attacker can exploit a remote or a physical intrusion depending on the type of access he has. If an attacker has physical access to a vehicle, he can have access to the Electronic Control Unit (ECU) of the car as well and take control of it.

Considering the tremendous benefits expected from vehicular communications and the number of vehicles around the world, it is also clear that the vehicular communications will become the most relevant ad-hoc networks. The more used a system becomes, the higher will be the number of threats that will arise. So it is essential to provide an efficient security system to prevent these threats. Some of the threats over VANET are described bellow as depending on the target of the attacker and how the attacker has access to the vehicle.

There are two different ways of gaining access to the vehicle: by physical access or by wireless methods.

In case of physical access to the vehicle, a mechanic or a bad-intentional person can plug a malicious component into the On Board Diagnostics (OBD) and make changes on the car software that might create malfunction of the system. Hacking devices like these can be used only once to perform the desired changes or be permanently attached to the car [23].

Wireless communication in cars can also be a threat and a way-in into the cars' system.

Nowadays each car is equipped with more than one wireless communication device as DSRC, Bluetooth or any other RF communication that can compromise the security of a car [23]. A CD player can also be a threat and it has been proved by Checkoway an his colleges in [24] that a specific song can affect the security of a car. The exploit here is when a firmware update can be made by special information on a CD as for example playing a specific modified song in Waveform Audio File Format (WAV).

2.5.1 Security Infrastructure

When security is needed in a certain place, a set of secure protocols are used to provide that the environment is securely safe. A secure infrastructure is thereby defined to provide the services and secure protocols that will grant a secure scenario for vehicular communication. A very well defined secure design of the infrastructure is needed, from cryptography algorithms to secure protocols. An overview of secure algorithms schemes and how they work was already introduced in section 2.4.

2.5.2 Security Challenges in VANET

VANET is a special and new type of networks and it will rise security threats and new requirements that are different from the ones we are used to deal with in a regular computer network. Some of these challenges are described below:

The risk attached to a Vehicular Network is a lot higher than in a normal computer when that is compromised. The hacking for a OBU can cause devastating physical security problems while in a computer the worst that can happen is a hard drive to get corrupted. It is therefore said that the safety applications will never take control of the vehicle, but they will

only provide vital information to the user [5].

VANET allows the creation of multiple services with financial profit to provide new maps, location based services, road tolling, digital infotainment and a lot more. Therefore the communications need to be secured in all types of data. In a home computer users do buy anti-virus and firewalls to prevent attacks, but in a vehicular environment drivers will not have to be worried if they do have anti-virus or if it has the last update. These processes then need to be somehow automatic to be completely transparent to the user.

Cars will be travelling from one place to another and the tracking of those should not be possible by anyone unless it's required by an authorized authority as the police. This way the driver's information as time, location, payments or any other should be kept in secret in a way that this information cannot be linked to a car and be traced. Privacy is then a big concern for VANET because police could for example issue speeding tickets based on the velocity that the beacon transmits.

2.5.3 Types of Attacks

The number of attackers that will try to take some advantage from these networks is likely to be significant, from crackers to hackers/academic hackers, they will be constantly trying to explore the network at theirs faults. Multiple types of attacks are possible: eavesdropping, message content modification and physical attacks. The main characteristics of the different types of attacks that may arise will be detailed bellow: [5]

- **Eavesdropping:** This is defined as the act of secretly listening to others' conversations, in this case scenario, an eavesdropper will collect several messages and will analyse them to gather the important information in his profit.
- **Message manipulation:** An attacker will inject in the network messages that were previously modified by him in order to transmit false information. This can be used to transmit false location and all other types of alerts (e.g as the occurrence of a fake accident).
- **Denial of Service (DoS):** The DoS attack is defined as the attempt to make a machine or network unavailable to its users. The way this can be achieved in VANET is by physically jamming the used frequencies or by injecting too many messages at the same time in a way that it will flood the communication channel. An attack like this is not considered a top risk attack because, in the worst case scenario, the level of security on the road will be the same as the one existing today.
- **Replay Attacks:** Replay attacks are based on the re-injection of a message in the network, a captured message can therefore be injected later in time or in a different location.
- **Physical Attacks:** These attacks are made when the attacker has access to the hardware, the purpose might be to gather secret keys and identification values that were introduced in manufacture time by the producers.

The use of cryptography algorithms and protocols as defined in the IEEE 1609.2 can be applied to prevent some of these types of attacks as the replay, eavesdropping or message manipulation. Regarding the physical attacks they can only be secured if secure hardware is produced to protect secret keys from being extracted.

2.5.4 IEEE 1609.2 Draft Standard

The IEEE 1609.2 D17 Draft Standard is the security standard in which the implementation of the security services will be based. The choice of using this standard aside from the European Security standard was because this is the most advanced automotive standard related with security. Its aim is to secure all the communications that are performed in a vehicular communication, defining which security mechanisms should be used and how the packets should be correctly formatted.

The required algorithms that the standard forces to be used in order to provide adequate security are the public key algorithms based on elliptic curves. The only symmetric algorithm that the standard refers is the AES-128 to be used with the ECIES public key algorithm. Therefore the standard leads to the following specification in terms of security algorithms:

- Digital Signatures using ECC over prime fields, ECDSA with NIST Fp curves.
- Encryption using ECC, ECIES.
- Hash Algorithms - SHA-1 and SHA-256.
- Symmetric scheme AES.

The ECDSA should support a 224-bit and a 256-bit implementation, the ECIES should work over a 256-bit key and the AES should work over a 128-bit key.

The standard also refers the creation of specific certificates called WAVE-Certificates which are more compact for performance reasons.

Although this is the most recent standard for security in VC it is very poor regarding its content, being still in Draft Version. A lot of topics are still open, by not specifying how certificates should be shared among elements on the road and what are the secure protocols to correctly use with the ECIES and AES algorithms. The main focus of this Standard is to allow authenticity by using the ECDSA algorithm.

2.5.5 Performance Requirements

In Vehicular Communications one of the crucial aspects towards deployment are the security requirements. The cost to equip a car with the state-of-art computer is too high so a reasonable alternative is to use embedded processors. Vehicular communications need to be secured so the security attached to these communications do not cause a significant overhead.

In VC cars are constantly interacting/checking each other (beaconing) for example to send information about their position or the weather conditions. These beacons are commonly sent at a rate between 1 and 10 per second which will cause some overhead in the network. Each

of these beacons carry safety information which needs to be signed (secured) by means of digital signatures and consequently verified by the receiver, for example a car may need to verify each $100ms$ all the beacons that come from nearby cars.

In DSA, the signature verification is the most consuming part of this security mechanisms. The sender has to generate one signature per message, and the receiver has to verify two signatures (message and certificate).

In a real case scenario if we want to estimate the number of signatures verification we must consider the worst case situation, for example a large highway during a traffic congestion. In [25] they state that "Assuming a neighbourhood density of 200 vehicles and beaconing rates between 1 and 10 per second, each vehicle needs to generate between 1 and 10 signatures per second and needs to verify between 400 and 4000 signatures per second." In this scenario and considering beaconing each $100ms$ we can reach up to 4000 signatures per second so summing up, the performance requirements are really high, there is no off-the-shelf cryptographic co-processor capable of such a signature verification load [25].

Some different approaches can be considered to achieve the security requirements:

- A fully software solution.
- Hardware solution.
- Smart Cards.

Software Implementation Approach

A OBU is equipped with an embedded processor, but the performance constrains are very high so even a fast processor as a Core i5 is not fast enough. Cryptography operations are performed over 224 or 256 bit operations which makes computation hard.

Benchmark results from the ECDSA with NIST P curves are scarce, but some results from a research found in [25] states that with a Intel Core i5 520M@2.4GHz achieved the following signature generation and signature verification results respectively: 250/s and 192/s. As referred before even with a core i5 processor, it is not possible to perform the extreme number of 4000 signature/second [25].

Smart Cards Implementation Approach

Smart Cards are a good alternative to a pure software implementation because they have a dedicated arithmetic unit for cryptography operations. Also they are designed to be tamper safe, being safer against physical attacks. Smart cards are designed to have dedicated long arithmetic cryptography modules that operate with words of 1408 or even more (eg. Crypto@1408 Smart Card). In table 2.2 it is shown a comparison of execution times of ECDSA with 256 bit length between two smart cards. It is important to notice that the clock frequency used in the smart cards (SLE88@66 MHz and SLE78@33 MHz) are very low compared with the i5 Core processor used in the software implementation [25].

Smart cards are not designed for full optimisation but mainly to have low power consumption, area and costs.

	SLE88@66 MHz	SLE78@33 MHz
Signature Generation Time in ms	9	98
Signature Verification Time in ms	16	54

Table 2.2: Smart Cards Timings [25].

Hardware Implementation Approach

As it was referred above a pure software implementation or even a smart card implementation do not satisfy the cryptography requirements for a Vehicular Communication System. An approach to this problem is to use Field-Programmable Gate Array (FPGA) and develop specific hardware implementations of a cryptographic processor. Full hardware implementations of ECDSA are scarce. In [26] the author proposed an implementation for a fully hardware version of the ECDSA algorithm with 163bit length. The achieved times for signature generation and verification were respectively $0.94ms$ and $1.61ms$. Another solution proposed in [27] designed to operate over prime fields with a key length of 256-bit, for a FPGA Xilinx Virtex 5 using 14256 LUT/FF achieved a $7.15ms$ generation time and a $9.09ms$ verification time. Implementations like these are still too slow for the worst case scenario, verifying only a few messages per second. Even with a fully hardware implementation of ECDSA a factor of more than 10 times is still missing. To help improve these efficiency requirements, most authors state that the use of Colbitz curves instead of NIST prime field curves could make the computation time a lot faster. Kimmo Järvinen, in [26] says that an improvement of almost 50% can be achieved with the use of these curves.

2.5.6 Related Work

Vehicular communications have been debated over the last few year as being the upcoming future technology for cars. This field has been a subject of study by manufacturers and universities to better understand and try to deploy a prototype that fulfils the VANET requirements.

The first projects were more interested in proving the feasibility of the system rather than being worried with higher layers, as applications or security. More recently projects from the US Department of Transportation, Rohde Schwarz SIT GmbH [25], the Secure Vehicular Communication (SeVeCom) [28] or other entities always refer the analysis of the security requirements and propose their security infrastructure. Projects like these play an important role in influencing some versions and reviews of IEEE 1609.2 Standard.

The *escrypt* company [29] already developed a sales product which implements the IEEE 1609.2 Draft Standard providing a highly optimized cryptographic engine to perform up to 400 signature generations/verifications over elliptic curves per second. It also provides other cryptographic algorithms as ECIES-AES-CCM as well as WAVE certificate handling.

A security analysis of Vehicular Ad Hoc Networks is described by Samara in [30]. Samara also talks about the security challenges, the requirements to achieve a secure system, and discusses previous proposed solutions from other authors.

2.6 Summary

After all the necessary background have been explained the proposed implementation of the security services for vehicular communications is presented in the next chapter. Although it is known from other researches as Papadimitratos in [31] and Rohde & Schwarz SIT GmbH project [25] that a pure software implementation is not capable of satisfying the security requirements of a VC, it was decided to do it.

The main purpose of this approach is to clearly understand how much computation power is needed to implement and provide the adequate security on VC. With this implementation it is expected to understand if security can be the bottleneck of a VC network.

In the next chapter an implementation of the security services focussing on the ECDSA algorithm as the core of security in VC is presented.

Chapter 3

IEEE 1609.2 Implementation

3.1 Introduction

In this chapter a software implementation of the security services is proposed. The choice was to implement in C programming language to clearly understand the performance requirements of pure software implementation in a Vehicular Communications (VC) scenario.

Security services defined in the IEEE 1609.2 D17 Draft Standard along with a cryptography engine were implemented with the help of OpenSSL as the library to perform the required cryptographic algorithms.

The choice to use the OpenSSL library was not arbitrary as it is a free and open-source C/C++ library for cryptographic operations and algorithms. It is not a cryptographic Application Programming Interface (API) but instead it contains all the required functions to implement the required algorithms.

Other software libraries as Crypto++ or BeeCrypt were considered but as OpenSSL was well documented and is widely used it became the preferable choice.

3.2 Architecture

The implementation architecture was divided into 3 main modules: a module that handles all the secure protocols, another one that is responsible for the cryptographic algorithms and another one containing all the secure keys and manufacture values. The module that deals with security protocols handles requests and interprets the responses from the cryptography engine. It is also responsible for the requests to the locally stored keys and provides them to the cryptographic engine. This module is the interface between higher layers (e.g applications) and the process of securing data. The private and public keys, along with certificates are stored in the module that contains all the manufacture values and secure keys. Additionally to these 3 modules a Global Positioning System (GPS) module was used to support the application by providing location and time information.

The proposed security model is based on the conceptual security model referred in [28] which is illustrated in figure 3.1.

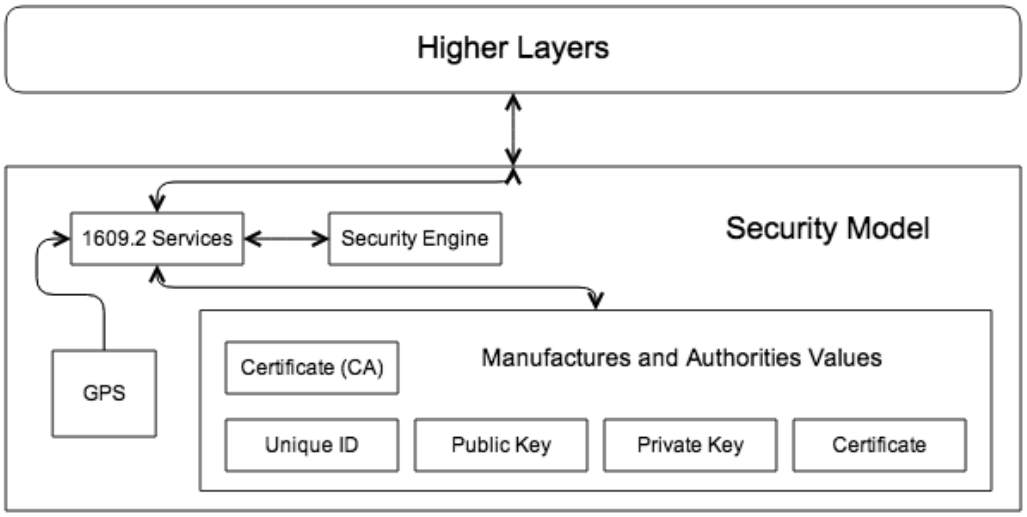


Figure 3.1: Security Model.

The engine which performs all the cryptographic algorithms was implemented with the OpenSSL library in order to provide all the specified algorithms in the standard. It is though capable of performing Elliptic Curve Digital Signature Algorithm (ECDSA) 224 and 256 [32], ECIES, Hash algorithms, certificate generation and verification. In figure 3.2 the engine with all the supported operations and building blocks is presented.

As referred before in chapter 2 the IEEE 1609.2 Draft Standard only defines secure protocols for interacting with the cryptography engine for the ECDSA algorithm, which from now on is the focus on this dissertation.

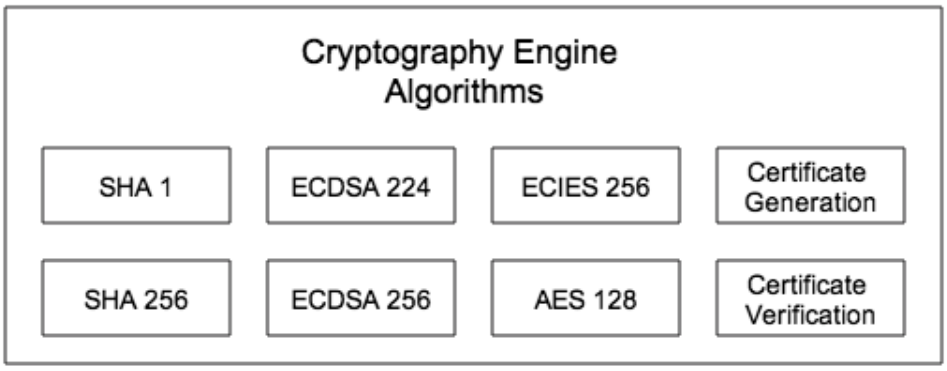


Figure 3.2: Cryptographic Engine with the required algorithms.

3.2.1 Required Software Resources

After the architecture is defined a set of software resources and libraries were used in order to implement it. Next they are described and an explanation of how they work and what their purpose are is presented.

Hash Table

An hash table is an abstraction of a common array which has the ability of having any type of reference as the array index. When we use an array, the index must be an integer that starts with 0 and goes over n. In the hash table the index can be anything we wish and it is called the *key*. The content of the array is called the *value*. So a hash table is a structure that has a *key* and a *value* in which the content can be accessed very fast by the *key* and it allows a fast search of unordered data.

Serialization

Serialization or more commonly marshalling is the process of transforming data structures into a buffer of data. This buffer of data is then suitable for storage or transmission. The reverse process is also possible, which allows a received buffer to be de-serialized and creates a data structure as the original one.

The data that needs to be signed according to the standard [21] is a structure, *ToBeSignedData*, which contains several data types. The signature of data needs to be done over a buffer of data, so the *ToBeSignedData* was serialized before being sent to the signature process. Serialization was also needed to transform the *1609dot2* packet structure into a buffer that was suitable to be sent over the network.

When serialization of a data structure contains variable data lengths, it means that an additional data field is required for each variable data. This extra fields are required to store the length of the variable data length, this is the only way to allow a correct de-serialization of data and it is able to reconstruct back the data structure.

Sockets

When the connection between two programs or two machines is needed, the use of a mechanism called *sockets* is used. This is a feature that most operating systems have and it allows a client-server connection to be used. Each time a socket is needed a *socket* API is used in the client and server programs to allow the establishment of a connection.

The way the interaction between Client-Server is done is by proceeding in the following way [33]: a server creates a TCP socket (*socket()*), binds it to a determined port (*bind()*), starts listening (*listen()*) and is ready to accept a connection from a client. The client also creates a TCP socket and establishes a connection. Both the server and the client are ready to communicate (*recv()* and *send()*) and share information. After the communication finishes the client closes the connection (*close()*) and consequently the server closes it too. After the connection is closed the server starts listening again for new connections from other clients.

The server only needs to have knowledge about the used port of communication and the client must know the server address and port.

GPSd

The *gpsd* is a daemon for GPS devices which interacts with various GPS brand devices through Universal Serial Bus (USB)/serial interface. The data from the GPS is acquired

through a TCP connection of the host computer and allows multiple access from different clients.

The purpose of the *gpsd* it to provide an open source interface for Linux users to easily interact with different GPS devices and obtain information as speed, location and more [34].

The *gpsd* was used in this implementation in order to get the time stamp and location to insert in the *1609dot2* secure packet. The IEEE 1609.2 D17 Draft Standard does not specify if location values should be obtained by the security layer or if a connection to higher layers must be performed to request a location. In this implementation it was decided to directly obtain the location and time-stamp values from the *gpsd* daemon.

The location structure should contain a latitude and longitude values represented by a int32_t value and encoded with precision 1/10th micro degree. The *gpsd* returns these values in degrees which were converted to be according to the standard.

3.2.2 Cryptographic Material

The keys and certificates that should be inserted into the On Board Unit (OBU) are referred as crypto-material and are stored in manufacture time. In this implementation it is assumed to have a fully Public-Key Infrastructure (PKI) deployed, meaning that the algorithms to share certificates and to manage certificate-revocations were not considered.

The crypto-material as the private keys, the public-keys and the certificates were stored in each of the cars. Each car has a 224-bit and a 256-bit key-pair length and a certificate which can be chosen depending on the required algorithm. The algorithms and processes to share certificates were not the focus of this dissertation so it was decided to include in each car all the other cars certificates.

This way each vehicle contains the following crypto-material:

- A private and public key associated with the vehicle.
- A certificate containing the vehicle public key.
- The certificates that belong to all other cars.

When a message is sent over the network, instead of containing the entire certificate attached to it, only a Hash of the certificate is sent. The SHA-256 is the chosen algorithm to perform this operation, but instead of adding the entire hash (256 bit) to the packet only the 8 less significant bytes are added [21]. Each car also contains a table of hashes, identifying all the stored certificates and its corresponding hash. Therefore when a message is received it contains the 8 less bytes hash of the corresponding certificate which is compared over the table of hashes and the corresponding certificate is loaded for verification.

An implementation like this implies a centralized system. When a new car enters the system, all the other cars in that area need to update their certificates data base in order to receive that new car certificate. This could be supported by the Road Side Units which could transfer and update the certificates data base of the cars when they enter a new area.

3.2.3 Data Flow

The way the signature generation/verification process works is defined by several services that handle multiple protocols. Different types of messages are going to be signed depending on the type of information that is going to be sent over the network.

Each time a connection is requested to the security services the flow of data is different depending on whether the request is to sign or to verify a message. In figure 3.3 a flowchart is shown in order to get an overview of how data is chained until the final result is obtained.

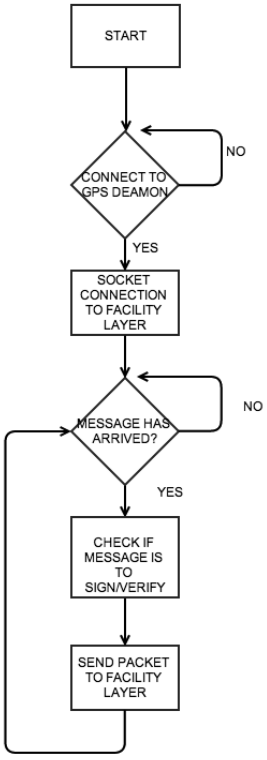


Figure 3.3: Overview of the implementation architecture flow.

The program starts by trying to establish a connection to the GPSd Daemon and after it is connected it creates a client/server connection with the facilities layer. From this point on the system stays in loop waiting to receive a connection from the client. As soon as a message arrives, its content is analysed and a process to verify or sign a message is performed. Then the message is returned to the facilities layer and the system stays again waiting for new messages to come.

The process to create the signature generation or signature verification of a message is detailed in figure 3.4 which describes how data is processed. There are two important functions (defined by the IEEE 1609.2 Standard [21]), the *WME SEC SIGN DATA* and the *WME SEC VERIFY DATA*, which are responsible to handle the protocols to sign or verify the messages correctly. It is in these functions that the access to the GPS data and the load of keys are made. In section 3.4 these protocols are explained in more detail.

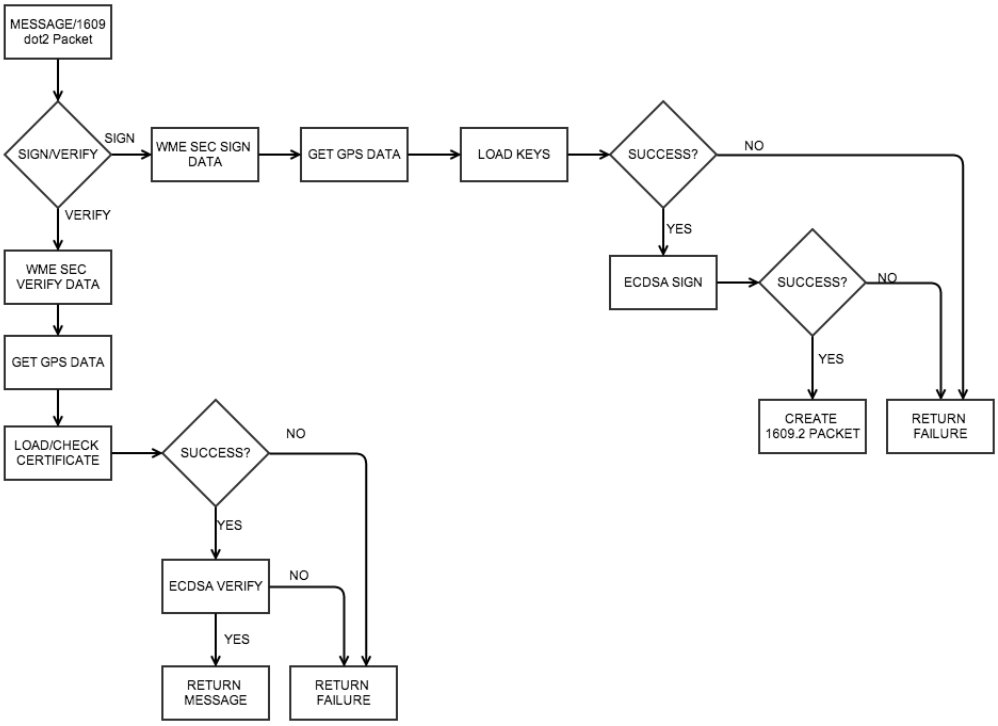


Figure 3.4: Overview of the implementation architecture flow.

3.3 ECDSA - Implementation

The ECDSA algorithms were implemented in C programming language and with the help of OpenSSL library [35].

The implementation of the algorithms and profiling of the code is going to be analysed to help understand the performance requirements for such an implementation in a real vehicular system.

As referred before, the main reference of the standard is that vehicular communications should support the ECDSA signature algorithms with the specified NIST Elliptic Curves P224 and P256. In order to analyse the implemented code at a deep level, a debugging, profiling and memory leak application was used. Valgrind [36] was the chosen one as it gathers all the necessary applications to analyse the code.

Signing and verifying messages is the purpose of this algorithm so the created application was developed to perform signing messages with different payload sizes and with the P224 and P256 curves. Verification of these messages was also made to get timings for verification and signing in different situations.

In figure 3.5 a flow chart was extracted from the profiling application Callgrind that is part of the Valgrind application. In this graph it is clear how much time is spent in each function to perform a signature generation or verification. A run of 100 signatures for each NIST curve and with payload size increasing from 100 characters to 200 characters was analysed to get the mean execution time in each function.

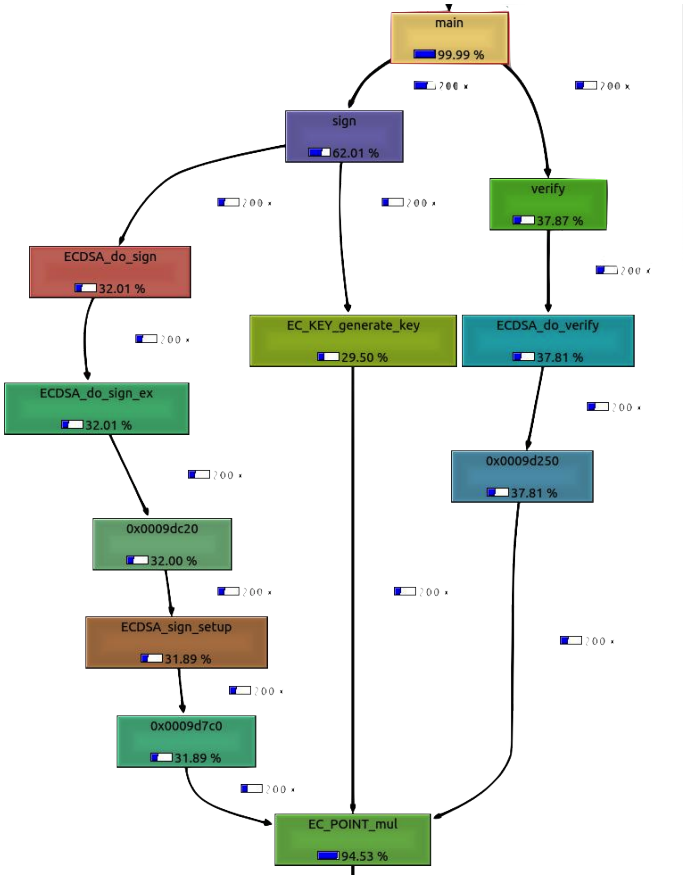


Figure 3.5: Callgrind Chart Flow.

The percentage in each block means the percentage of time that each function takes in its operations, regardless of the computer it is running on.

The signing process takes more time than the verification process (62.01% and 37.87% of the time respectively) as can be seen in figure 3.5. This happens because in this implementation, for each run, the signing process needed to generate a key pair, generate the random data and perform the signing process itself.

A further analysis in chart 3.5 shows that the verification process itself is computationally harder than the signing one. The OpenSSL functions to sign (ECDSA_do_sign) and verify (ECDSA_do_verify) take 32.01% and 37.81% of the time, respectively. This was the expected result according to the theory of cryptography, as the verification process has more operations than the signing one [26].

3.3.1 Open-SSL API

In this section the OpenSSL library used to perform the required cryptographic algorithm with the specific curves will be explained. With this library it is possible to implement almost all known cryptographic algorithms, but here only ECDSA is explained in detail.

Key Generations

The key generation process is not ECDSA specific and it is generated in the following way [35]:

```
EC_KEY *eckey=EC_KEY_new();
if (eckey == NULL) { /* Handle error */ }
else {
   EC_GROUP *ecgroup= EC_GROUP_new_by_curve_name(int nid);
   if (NULL == ecgroup) {/* Handle error */}
   else {
      int set_group_status = EC_KEY_set_group(eckey,ecgroup); /* returns 1 for
          success. */
      if (set_group_status != 1) { /* Handle error */ }
      else {
         int gen_status = EC_KEY_generate_key(eckey); /* returns 1 for success. */
         if (gen_status != 1){ /* Handle error */}
           } } }
```

The *nid* value that is assigned in the function EC_GROUP_new_by_curve_name(int nid) is the name of the curve that is used to create the Elliptic Curve Group, which in this case are the NIST prime curves defined respectively by the following names: *NID_secp224r1* and *NID_secp256k1* [35].

The OpenSSL library allows the use of multiple curves and its list can be viewed by running the following command on terminal.

```
# openssl ecparam -list_curves
```

The *EC_KEY* eckey* is a structure that is composed of parameters that define the type of key generated along with the private and public key. It's structure is defined bellow, and when the EC_KEY_new(*void) is called it is created the following way [35]:

```
        eckey->version = 1;
        eckey->group  = NULL;
        eckey->pub_key = NULL;
        eckey->priv_key= NULL;
        eckey->enc_flag= 0;
        eckey->conv_form = POINT_CONVERSION_UNCOMPRESSED;
        eckey->references= 1;
        eckey->method_data = NULL;
```

The ecgroup value is changed with EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group) function according to the group curve used. The private and public keys are assigned

when the int EC_KEY_generate_key(EC_KEY *eckey) function is called [35].

Signature generation

For the signature generation, considering that the private key is already generated, and apart from the data processing it is only needed to call the following Open-SSL function [35]:

```
ECDSA_SIG * = ECDSA_do_sign(const unsigned char *dgst, int dgst_len, EC_KEY *eckey);
```

This function has as return value the signature that was generated for the input data.

As discussed before a signature has twice the size of the key length used in the ECDSA algorithm, so the ECDSA_SIG is a structure composed of two BIGNUM:

```
struct
{
   BIGNUM *r;
   BIGNUM *s;
}ECDSA_SIG;
```

If the hexadecimal representation of the signature is needed the following function can be used to print both values.

```
char *BN_bn2hex(const BIGNUM *a);
```

Verification

The verification of a signature is composed of several steps that must be taken into consideration to make a verification with OpenSSL.

```
EVP_PKEY* pk = EVP_PKEY_new();
EC_KEY* publickey;
pk  = X509_get_pubkey( X509 );
publickey = EVP_PKEY_get1_EC_KEY(pk);
int ECDSA_do_verify(const unsigned char *dgst, int dgst_len, const ECDSA_SIG *sig,
    EC_KEY* eckey);
EVP_PKEY_free(pk);
EC_KEY_free(publickey);
```

The return value from the *ECDSA_do_verify* functions determines if the verification was a success (return value = 1), failure (return value = 0) or an error occurred (return value = -1) [35].

3.3.2 Certificates

The certificates used in a vehicular environment must follow the rules defined in the standards. Since in this dissertation the WAVE protocol stack was followed, the IEEE 1609.2 Standard defines the WAVE-Certificates which are a special type of certificates for Vehicular Communications. As a PKI was not implemented, the used certificates were generated using OpenSSL which are a standard certificate type. These certificates are larger compared with the expected size of a compact WAVE-Certificate. The used certificates are encoded in X.509 which is a specific format for certificates. Their size can vary, depending on the key length used for the public key algorithm, on the size of identification and on some optional values. In this implementation the used certificates had approximately 956-bytes (containing the public key, signature, validity and other information about the user) which are too big compared with the expected WAVE Certificate size of about 120-bytes as refereed in [5].

An application to generate private and public keys together with the respective certificate was implemented separately to support the main application that makes use of this cryptographic material.

3.4 Implementation of Secure Protocols

So far the cryptographic engine supported by OpenSSL libraries is able to perform all the required algorithms from the standard. As already mentioned, a cryptographic engine is not enough if it is not strongly supported by some protocols.

The IEEE 1609.2 D17 Draft Standard defines these protocols to use with the ECDSA algorithm. These protocols are handled by the Wave Management Entity (WME), which is responsible for the handling of data from higher layers. The WME also makes requests to the security module to ask for secure data. After data gets secured it needs to be sent over the network, a secure packet structure is defined by the IEEE 1609.2 as follows:

A secure packet is referred in the standard as a *1609dot2* packet. Its structure is composed of multiple sub-structures as it is shown in figure 3.6.

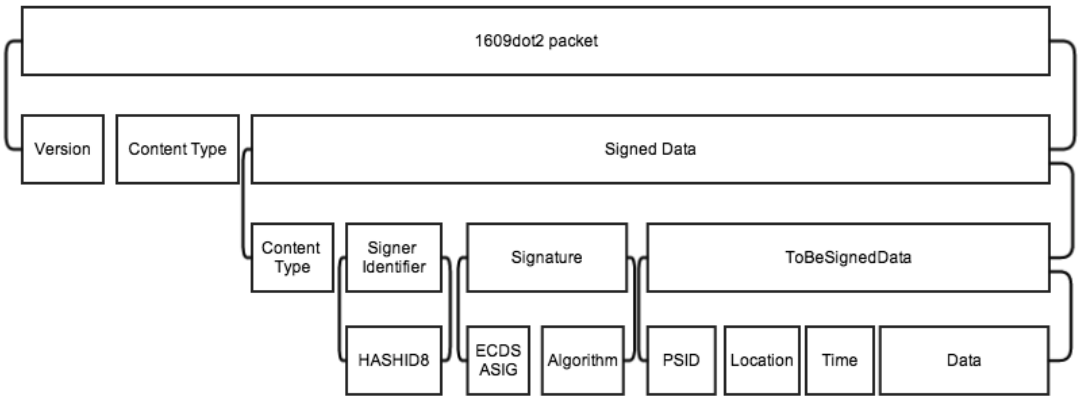


Figure 3.6: 1609.2 Secure Packet Structure.

- **Version** - Defines the version of the implemented 1609.2 standard;
- **Content Type** - Identifies if the packet is unsecured, signed or encrypted;
- **Signed WSA/WSM** - The signed data itself;
- **ToBeSignedData** - The data that is going to be signed;
- **Signature** - The signature of the respective data;
- **Data** - The message itself;
- **Time** - The time-stamp in which the data was signed;
- **Location** - The location where the data was signed;
- **ECDSA Signature** - The signature itself;
- **Algorithm** - The type of algorithm used to sign the message;
- **HashID8** - The less 8 significant bytes of the SHA-256 of the certificate.

The size of each field is only fixed for the Version, Content Type, Time, Location, Algorithm and HashID8, all the other fields vary depending on the size of each message. The version takes the number 2 as it refers to the version of the standard IEEE P1609.2TM/D17 and has the size of 1-byte [21]. Content Type and Algorithm also take 1-byte each. Time has a 4 byte length and Location has a 2D Location with latitude and longitude defined as a four byte length each.

The size of the signature also varies between 2 values depending on the signature algorithm used. As the standard refers, the implementation should provide 2 signature algorithm, an ECDSA with 224-bit and 256-bit length. The signature size has twice the length of the key length of the signature algorithm, which means that the signature can get a length of 56 or 64 bytes.

The size of all packet can then vary from 83 to 91 bytes without the size of the message. These messages can also vary in size depending on the type of messages and on the number of fields it carries.

In this implementation the type of messages that are going to be secured are the CAM messages. This messages are defined later in chapter 4 and considering only its mandatory values its minimum size should be around 54-bytes. This way the *1609dot2* packet should carry most of the time at least between 137 to 145 bytes.

Now that the secure packets were described the signature generation and the signature verification protocols are presented:

Process of a Signature Generation:

- Get the data to be signed;
- Get time and current position;

- Create a structure called *ToBeSignedData*;
  - . Data that is going to be signed;
  - . Generation time and location;
  - . Expiry time;
- Encode *ToBeSignedData* structure into an octet string;
- Digitally sign the generated octet string with the ECDSA algorithm;
- Create another structure called *SignedData*;
  - . Type of data;
  - . The data;
  - . The signature;
- Create a *1609dot2* structure ready for transmission;
  - . Protocol version;
  - . Type of data;
  - . Encoded *SignedData*.

If the signature generation occurs as expected, the type of data in the *1609dot2* packet should be set equal to *signed data* to identify that the content is signed.

Process of a Signature Verification:
Get the *1609dot2* packet and make the following verifications. In case any of them fails, the message must be discarded.

- Decode the message and parse it into its structure fields;
- Check if protocol_version is equal to 2;
- Check if SignedData.type is equal to signed;
- Check if expiry_time is later than generation_time;
- Construct a chain from the received certificate to a known root certificate;
- Verify the certificate signature;
- Verify the signature of the message;

If none of these verifications fail, the message is correctly verified and can be passed to the higher layers which requested the verification (calling application).

In figure 3.7 the process of securing a message is presented, both for a signature generation and for a signature verification.

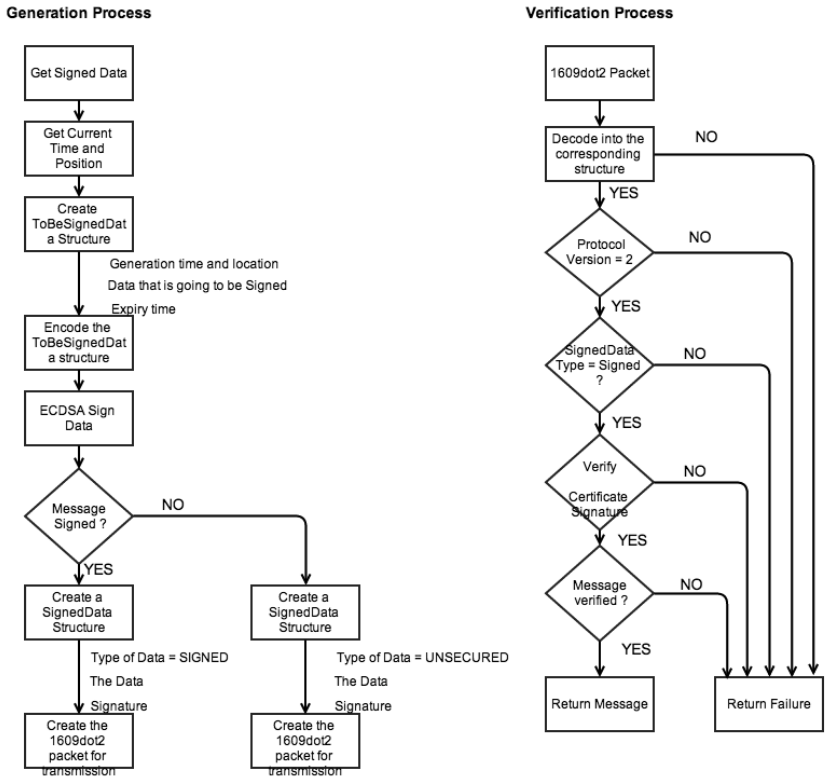


Figure 3.7: Signature generation and verification protocol.

3.5 Summary

In this chapter it was explained how the security services were implemented.

An architecture to provide the adequate security based on the IEEE 1609.2 D17 Draft Standard [21] was proposed. This implementation was purely developed in C programming language to understand the necessary computational power.

Next other modules were integrated to create a full-blown architecture that is capable of generating messages, secure them and communicate with the IT^{2}S Platform.

Chapter 4

Integration with WSMP and Facilities Layer

4.1 Introduction

After the security services have been developed there was the need to integrate this work with other applications as the WAVE Short Message Protocol (WSMP) and the facilities layer in order to have a full system working. All these applications/modules were developed separately by different persons in the scope of other dissertations, which made the integration a complicated task. From these 3 modules, the WSMP (developed by Paulo Sousa) and the security services were developed based on the WAVE Stack while the facilities layer (developed by Daniel Ferreira) is defined in the ETSI ITS Stack. The reason for having mixed these two stacks was due to the lack of standards in the ETSI ITS Stack which does not define any security standard for VC. Therefore a way to interact between layers and to achieve the desired objective was developed.

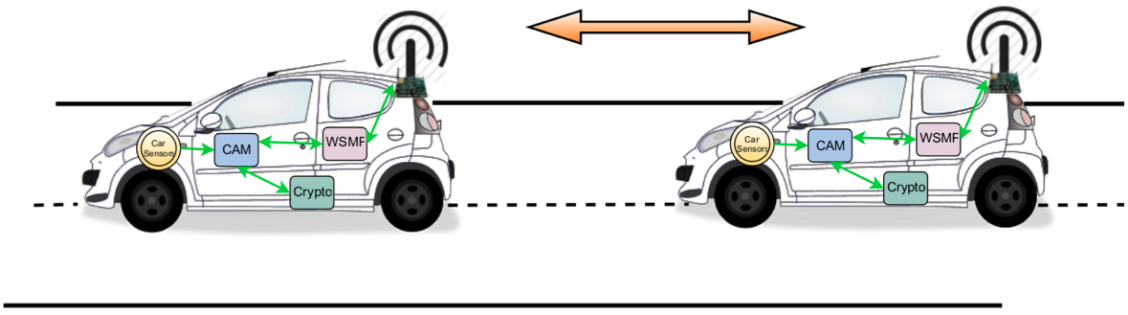


Figure 4.1: Communications between vehicles.

In this chapter all the modules will be described separately explaining how the integration and the communication between them was performed. The main purpose of this integration

was to have an architecture capable of generating messages, secure them and communicate through the Dedicated Short Range Communications (DSRC) platform IT^{2}S developed in the IT (Telecommunications Institute).

4.2 Overview of the Integration Architecture

The integration of all modules can be seen as a "black box" that generates messages and communicates with the IEEE 802.11p platform through the WSMP. Each module was implemented separately and the communication between them was made using socket communication. In figure 4.2 this "black box" is presented.

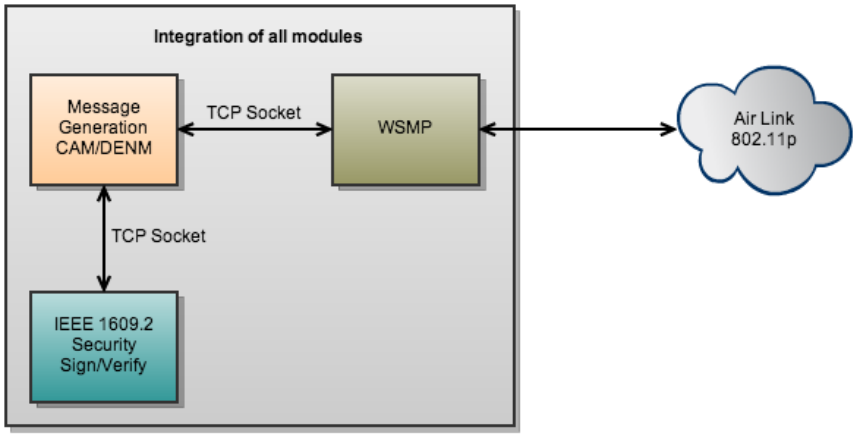


Figure 4.2: Top Level architecture of the modules integration.

A Cooperative Awareness Message (CAM) message is generated by the facilities module with all the information gathered from the vehicle: speed, location, direction and all types of valuable information. This message is sent to the security services which receives the message and digitally signs its content with its private-key. After the message gets secured in the format of a *1609dot2* packet, it is sent again to the facilities module which forwards its content to the WSMP. The WSMP generates the WSM packet and puts in its data field the *1609dot2* packet received from the facilities module. In figure 4.3 we can better understand how the flow of data is handled.

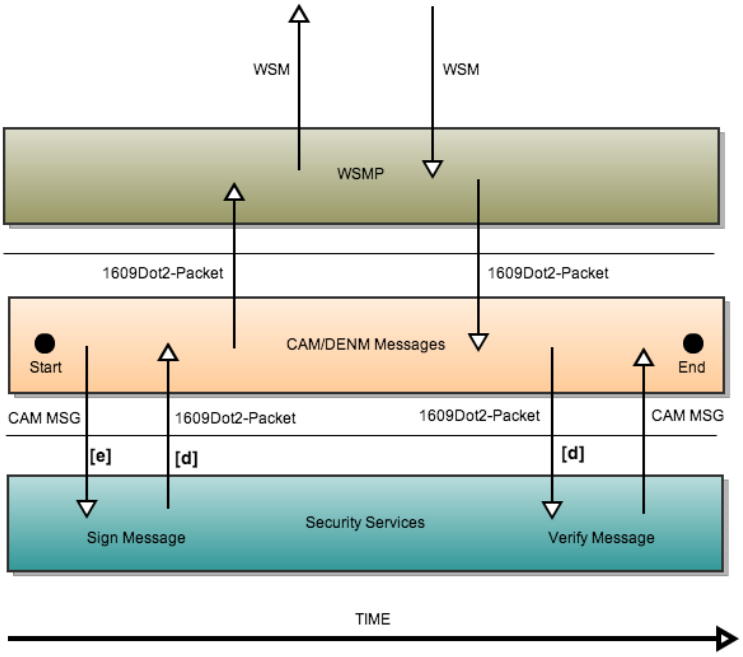


Figure 4.3: Architecture data flow.

The WSMP, Facilities layer and Security are complex modules that are going to be described in detail in the next sections.

4.3 Facilities Layer

The facilities layer is responsible for handling the generation of messages. In order to correctly generate these messages they need to obtain some specific information: current position, time-stamp and other valuable information from the car. This data is gathered from the Global Positioning System (GPS) or the On Board Diagnostics (OBD) sensors. From the European standard [37] two types of messages can be generated: the CAM and the Decentralized Environmental Notification Message (DENM).

4.3.1 CAM

Vehicles need to be constantly providing information about themselves (position or status of the vehicle) to other cars through the CAM. The main purpose of these messages is to provide useful information about all surrounding vehicles and for example, to alert drivers of an emergency vehicle approach.

These messages are time and position triggered, which means they are constantly being sent between a frequency of 1Hz to 10Hz and whenever there is a difference in position bigger than $5meters$ [37].

4.3.2 DENM

The DENM are more specific messages that are only triggered on special events. These messages are very specific and should cover a certain area within its generation location. Therefore they are multi-hop messages that can be re-transmitted from/to other vehicles [38].

The type of events that can cause the generation of a DENM message goes from an emergency breaking warning to the detection of a strong wind condition. In figure 4.4 a list of some events that cause the generation of these messages is shown [38].

Table 1: Triggering and termination conditions of DENM sending

Use case	Triggering condition	Terminating condition
Emergency electronic brake light	Hard breaking of a vehicle	Automatic after the expiry time
Wrong way driving warning	Detection of a wrong way driving by the vehicle being in wrong driving direction	Vehicle being in the wrong way has left the road section
Stationary vehicle - accident	e-Call triggering	Vehicle involved in the accident is removed from the road
Stationary vehicle - vehicle problem	Detection of a vehicle breakdown or stationary vehicle with activated warnings	Vehicle is removed from or has left the road
Traffic condition warning	Traffic jam detection	End of traffic jam
Signal violation warning	Detection of a vehicle being violating a signal	Signal violation corrected by the vehicle
Road-work warning	Signalled by a fix or moving roadside ITS station	End of the roadwork
Collision risk warning	Detection of a turning collision risk by a roadside ITS station	Elimination of the collision risk
	Detection of a crossing collision risk by a roadside ITS station	Elimination of the collision risk
	Detection of a merging collision risk by a roadside ITS station	Elimination of the collision risk
Hazardous location	Detection of a hazardous location	Automatic after the expiry time
Precipitation	Detection of a heavy rain or snow by a vehicle (activation of the windscreen wrappers)	Detection of the end of the heavy rain or snow situation
road adhesion	Detection of a slippery road condition (ESP activation)	Detection of the end of the slippery road condition
Visibility	Detection of a low visibility condition (activation of some lights or antifog)	Detection of the end of the low visibility condition
Wind	Detection of a strong wind condition (stability control of the vehicle)	Detection of the end of the strong wind condition

Figure 4.4: DENM Trigger Event List [38].

4.4 WSMP Layer

The WSMP was developed to work in a Linux based architecture and it was deployed in a daemon process in which communication primitives are supported. The communication between the network layer and the lower layers is maintained by a Raw Socket, while communication between higher layers is performed through TCP/IP Sockets.

The architecture of the implementation is presented in figure 4.5. This architecture is suitable for any Non-IP protocol and it can support both the WSMP and FIX Adapted for STreaming (FAST) Protocol, but for these tests the WSMP was used.

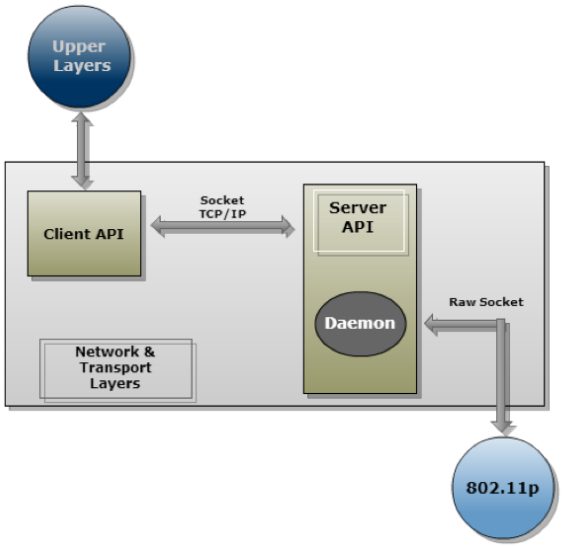


Figure 4.5: WSMP Architecture.

4.5 Interaction between Layers

After having explained all the modules, it is time to focus on the information flow between those modules providing details about the connections.

Each time a CAM or DENM message is generated it is inserted into a queue in order not to lose any message. The Message Handler is therefore responsible for the extraction of messages from the queue and sends them to the security services or directly to the WSMP.

The CAM Handler, the queue and the message handler are separated processes that are handled with threads to allow the correct function of the system.

In the reverse process, when a message comes from the network (WSMP) and needs to go up to the application layer, the message handler is also responsible for it and always forwards the message to the security services. The message is then verified by the security services and if it succeeds, it will be sent to the receiving application or, in case of failure, the message will be discarded.

In figure 4.6 it is clear how the connection between modules was performed.

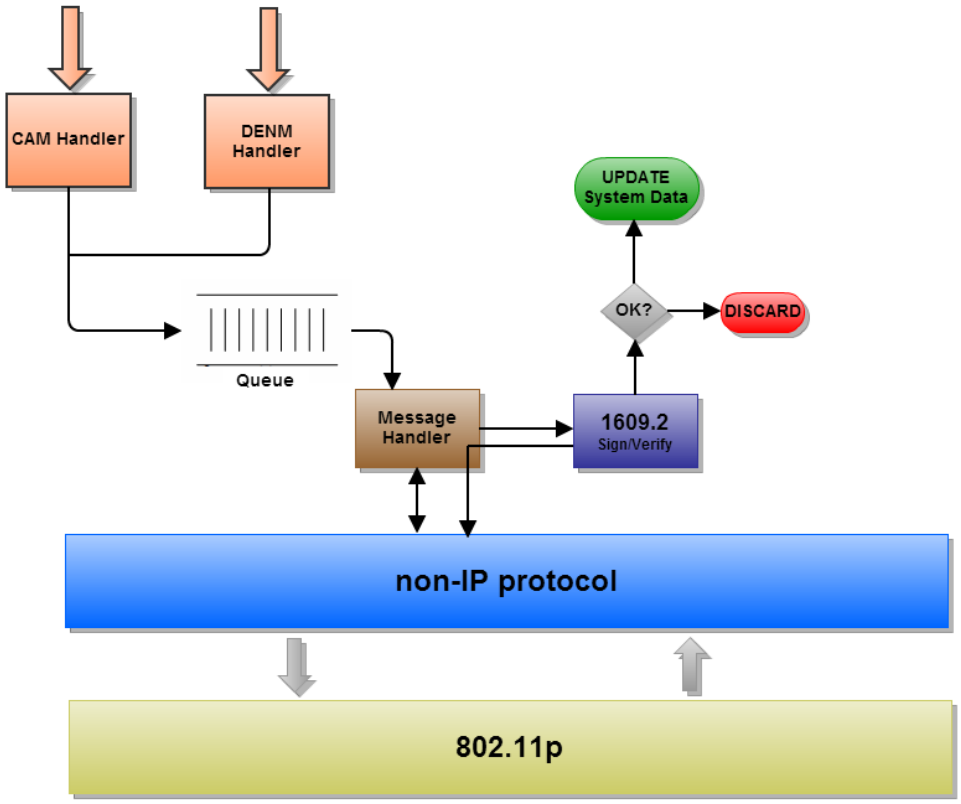


Figure 4.6: Overview of how modules were integrated with each other.

4.5.1 Facilities Layer - Security Services

As stated before, specific interfaces and a communication protocol had to be created to interconnect these modules and allow the communications between them.

Once a message is generated, it is sent over the security services to get the corresponding security level. In order to identify if the received message is to sign or to verify, an header had to be added to create a communication protocol between these two modules. This header is 1-byte long to identify if the message is to sign, verify or if there was an error in the signature generation or verification. This way each time a message is generated an header with the character $'e'$ is added signalling that the message must be signed. If the message gets correctly signed, this header is replaced by the security services with the character $'d'$, which means that it was correctly signed.

The packet is transmitted to the network with this header attached to the *1609dot2* packet. When the facilities module receives the packet, it checks the header and if it has the header $= 'd'$ it takes it out and only forwards the *1609dot2* packet to be verified by the security services. A message gets verified or not and notifies the facilities layer by sending the verified message back to it or by sending back the character $'n'$. It is now the facilities layer function to keep or discard the message in case of success of failure verification. In figure 4.3 it is clearly shown how the header is used in the message flow.

This communication protocol had to be used because at the moment of the implementation there was no defined standard to deal with the transference of data between layers and also because the layers were from different stacks.

The communications between the two modules are made by using sockets in blocking mode, meaning that when a message is sent to the security services the connection stays blocked until it receives a response, be it of success or failure.

4.5.2 Facilities Layer - WSMP

The communication between the facilities layer and the WSMP is performed in a server/-client connection methodology. The server (wsmp *deamon* process) is the first program to start running and waits for the clients to connect to it. The server must create a stream socket, assign a port and an interface to which the clients will connect through. The client (facilities application) tries to connect to the defined port and to the name server and if the server is listening in the same port the connection will succeed. A handshake is performed to associate the client with a PSID and after this the connection is ready to send or receive messages.

4.6 Experiments and Tests

The integration was performed in several steps, before getting to the final stage of the system. The communication among modules was performed using sockets and in the first approach each of the modules was running separately in a different machine. The security services, the WSMP and the facilities application were running in different laptops (figure 4.7). Then the modules were joined in the same machine in order to simulate the system as if it was running in a real On Board Unit (OBU) (figure 4.8). In this second approach the communication was still done by means of sockets but now the connection between modules was performed in localhost.

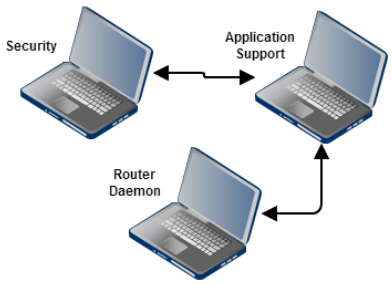


Figure 4.7: First approach to the final system.

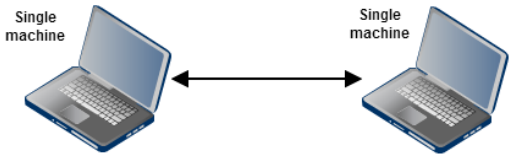


Figure 4.8: Second approach to the final system.

The connection between OBU was performed through Ethernet to simulate the connection through the IT^{2}S platform. This approach had to be used because it was dependent on other projects which were running at the same time, namely the implementation of the device driver for the IT^{2}S Platform.

4.7 Implementation on IT^{2}S Platform

In figure 4.9 the IT^{2}S board implements all the lower layers required by the ETSI ITS Stack to work as DSRC module. As referred in [39] "The IT^{2}S board comprises all the components required to implement the lower layers (IEEE 802.11p PHY and MAC time critical operations) of the ETSI ITS G5 protocol stack".

Figure 4.9: IT^{2}S Board Description [39].

This "black box" (the security services, the WSMP and the facilities application) is used on top of this platform to provide a way to generate, secure and transmit messages according to the standards.

After having performed all the experiments in laptops the migration to a more suitable environment for real tests was needed. Therefore a *Raspberry-Pi* was the choice to be the embedded single board computer that binds the higher layers with the lower ones. The configurations and management of the platform is achieved by a Universal Serial Bus (USB) connection with the *Raspberry-Pi*. As soon as the device driver for the IT^{2}S Platform is ready, the migration to this interface should be smooth and everything should work as expected.

In figure 4.10 there is a general overview of the whole architecture, from the lower layers(the IEEE 802.11p PHY and MAC), to the higher ones (Smart Phone).

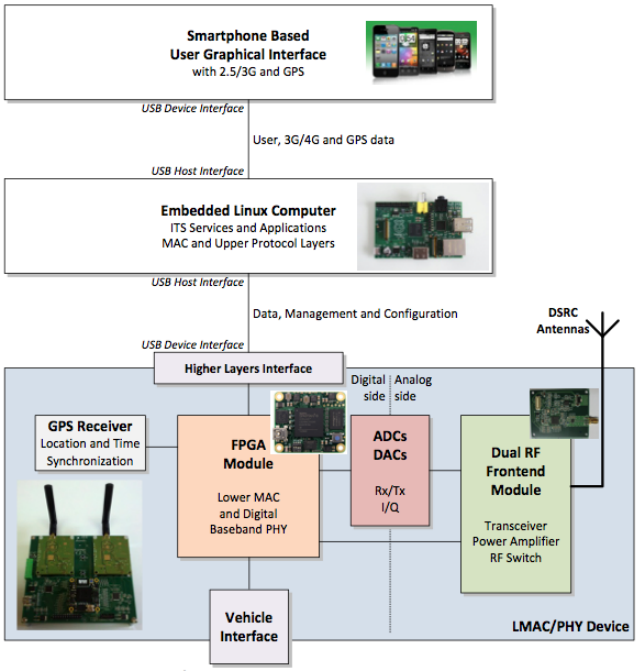


Figure 4.10: IT^{2}S Target Architecture [39].

4.8 Summary

In this chapter it was explained the integration of the security services with the WSMP and the facilities layer (generation of messages) in order to achieve a full-blown architecture. This integration was developed using different stacks which turned to be a difficult task. Therefore specific protocols had to be created to handle the interaction of data between layers.

In the next chapter the results obtained from the implementation will be presented.

Chapter 5

Experimental Results

5.1 Introduction

In this chapter the results of the security services and the cryptography algorithms defined in the IEEE 1609.2 D17 Draft Standard are presented. Timing values and how system responds to different payload sizes is evaluated. The integration with the WAVE Short Message Protocol (WSMP) and the facilities application is also evaluated both in a conventional laptop and in a Raspberry-Pi to simulate a real scenario.

In order to benchmark the system, several tests were carried out. The main focus of these tests was to benchmark the overall system that provides authenticity with Elliptic Curve Digital Signature Algorithm (ECDSA). In a first approach the system was benchmarked in a laptop and later in a Raspberry-Pi which was the option as the on-board computer for a real vehicular communication system.

The choice to use two computers in the experiments was to clearly understand how much computational power might be needed to achieve a good performance of the system. The power of both computers was very different which caused a significant difference in timing analysis. Table 5.1 provides a hardware and software comparison between the two used machines.

The following set of experiments were analysed:

- ECDSA algorithm timing analysis on Laptop with increasing random payload;
- ECDSA algorithm timing analysis on Raspberry-Pi with increasing random payload;
- Security Implementation timing analysis on Laptop with CAM Messages as payload;
- Security Implementation timing analysis on Raspberry-Pi with CAM Messages as payload;

5.1.1 ECDSA Timing Performance Analysis

For the analysis of the execution times regarding the ECDSA algorithm an application capable of signing and verifying messages without all the overhead caused by the security services was developed.

Specifications	Personal Laptop	Raspberry Pi
CPU	Intel Pentium Processor T2310 (1M Cache, 1.46 GHz, 533 MHz FSB)	700 MHz Low Power ARM1176JZ-F Applications Processor
Instruction-Set	32-bits	32-bits
Memory	3GB	512 MB SDRAM
OS	Ubuntu 12.04	Arch-Linux

Table 5.1: Hardware comparison between personal laptop and Raspberry-Pi.

In order to calculate the execution time of the algorithm to be evaluated, a processor tick timer was inserted in the code to count the number of ticks that each function runs. This way the exact timing of specific parts of the code could be analysed in detail.

Algorithm 5 Time Counter

Require: #include "time.h";

Require: struct timespec t1,t2;

Require: double dt;

1: clock_gettime(CLOCK_MONOTONIC, &t1);

2: Code for timing analysis goes here ...

3: clock_gettime(CLOCK_MONOTONIC, &t2);

4: dt = (t2.tv_sec - t1.tv_sec) + (double)(t2.tv_nsec - t1.tv_nsec) * $1e^{-9}$;

5: Runtime of code: dt;

Then the execution of the code was analysed for the ECDSA NIST curve P224 and P256 with the payload varying from 10-bytes to 2000-bytes. Within each payload the code runs 1000 times in order to calculate the mean execution time for the given payload size. The payload was randomly generated, containing only alpha-numeric values.

In graph 5.1 there are the execution times tested on a laptop and some conclusions can be taken:

- It is noticed that the cryptographic engine has different computation times depending on the size of the algorithm key.
- The differences between signing and verifying are also very explicit, as the verification and the signature timings are quite distinct. As mentioned in the literature [26], the verifying process is computationally harder than the signature process which is consistent with these values.
- Another interesting fact is that the signing and the verification process do not depend on the size of the payload. The reason for this is that it is only considered the arithmetic

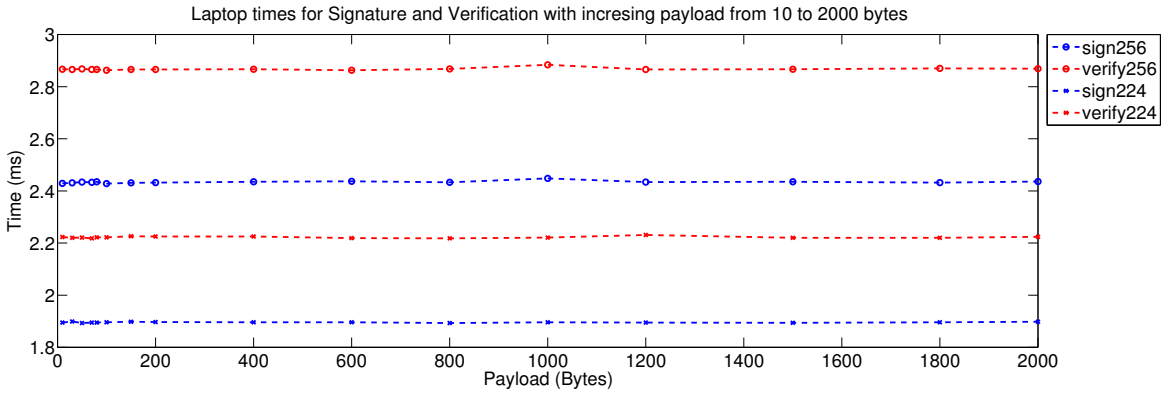


Figure 5.1: ECDSA timing on Laptop with increasing payload from 10 to 2000 bytes.

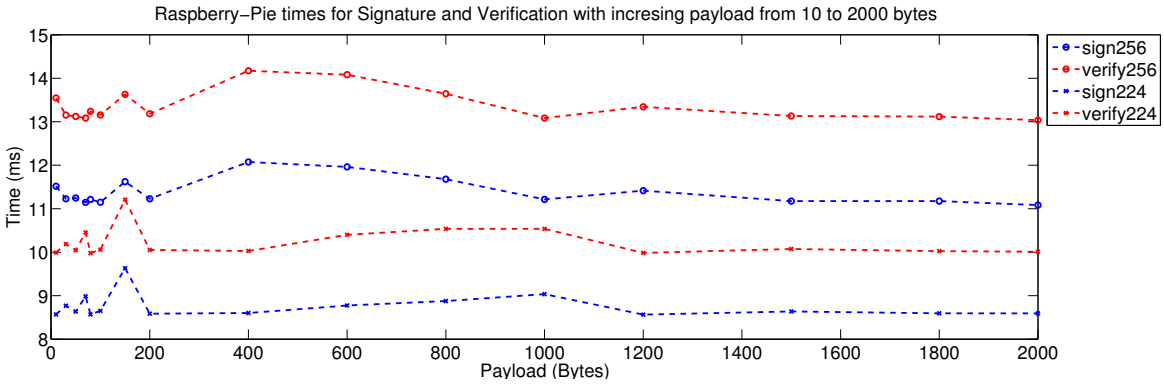


Figure 5.2: ECDSA timing on Raspberry-Pi with increasing payload from 10 to 2000 bytes.

operation to sign and verify and not, for example, the hashing of the payload.

Figure 5.2 refers to the computation times in the Raspberry-Pi. The execution times either to sign or to verify messages are higher comparing with the laptop execution times in figure 5.1. These values were expected as the hardware is a lot slower than the personal computer. Another interesting fact is that the timings either in signature generations or verifications are fluctuating a bit. This might be caused by the low memory and low power CPU that the Raspberry-Pi has.

Now comparing the overall timings in graphs 5.1 and 5.2 the Raspberry-Pi is about 5-times slower than the laptop.

In table 5.2 a summary of the mean execution time for both key algorithms and computers is presented.

It is very important to refer that all these values only represent timings for the OpenSSL signature generation and verification functions. An ECDSA Signature or Verification to be completed requires that a key-pair must be previously generated or the keys must be loaded and a hash function applied to the payload, for the OpenSSL functions to sign and verify to

Computer	Algorithm	sign[ms]	Signature/s	verify[ms]	Verification/s
Laptop	ECDSA P256	2.4339	411	2.8676	349
Laptop	ECDSA P224	1.8958	527	2.2222	450
Raspberry-Pi	ECDSA P256	11.3833	88	13.3581	75
Raspberry-Pi	ECDSA P224	8.7552	114	10.2238	98

Table 5.2: Table with mean execution times for ECDSA 224 and 256 for both computers.

be applied.

These values do not represent the overall system execution times but they were taken to benchmark the OpenSSL library.

5.1.2 Integration of CAM, WSMP and Security

It is now time to give an analysis of timings with integration of WSMP, the facilities application and security.

This step is important to better understand how the system handles the increase of the payload, the time for the whole process, the signature generation, the signature verification, the loading certificates and the hashing of data.

These experiments were both tested on a laptop and on the Raspberry-Pi to see the differences in performance (table 5.1).

System Benchmark with random data as payload

For this specific performance test only the laptop was considered to benchmark the whole system, increasing the payload of the messages with random data.

In figure 5.3 the process to sign and verify a message with different payload sizes is analysed and the computation times are presented. For each payload the values presented are mean values calculated over a run of 500 times of the code. The ECDSA algorithm considered was the ECDSA-256.

This experiment led to the following mean execution times for both signature generation and verification, $4.8128ms$ and $5.6003ms$, respectively regardless the size of the payload.

From the graph in figure 5.3 as the payload increases also the time either to sign and verify increases. This increase is sharpest in the signature generation while in the verification process between the 800 and 1200 byte payload there is a significant decrease.

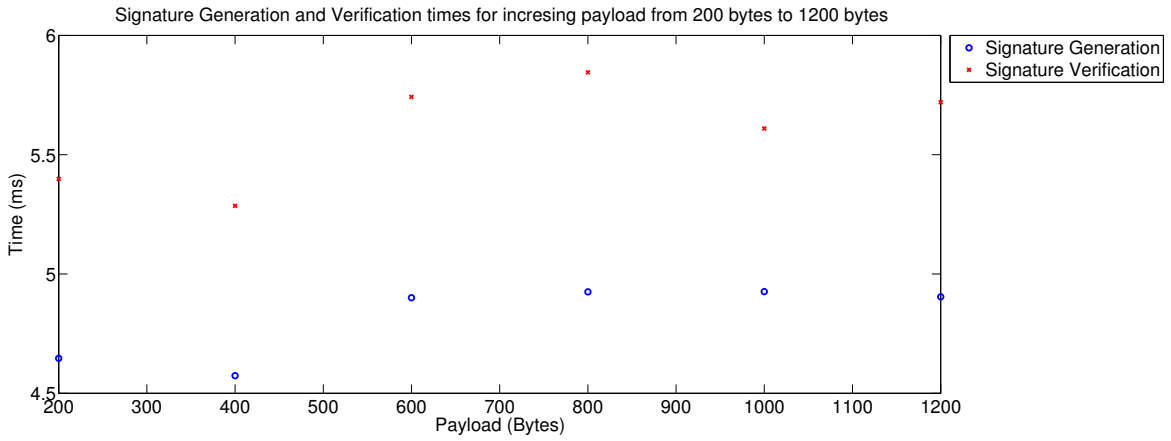


Figure 5.3: Entire system times for Signature and Verification.

Results Analysis with CAM messages as payload

Here it is considered a real scenario test bed with all the modules integrated and with the Cooperative Awareness Message (CAM) messages being transmitted at the rate of 10Hz. Ten thousand messages were sent at this rate to analyse the system performance. Within these 10000 messages two different sizes were generated, one with 53 bytes and another one with 67 bytes.

The experiments were performed again in two computers, a laptop and a Raspberry-Pi.

Laptop

The following results presented on table 5.3 were obtained from one of the computers that was running the whole integrated system.

Summary	
Total number of messages to sign	10000
Total number of messages to verify	9958
Total number of verification OK	9859
Total number of verification Failed	99
Percentage of failure (%)	0.99
Timings	
Signatures[ms]	3.850
Verifications[ms]	4.415

Table 5.3: Summary of program execution.

From the summary shown on table 5.3 some conclusion can be taken: all the messages were successfully signed while in verification the percentage of failure was close to 1%. These failures in verification happened because the WSMP arbitrary lost some bytes during the transmission of the packets causing the ECDSA verification to fail. Another important reference is that the number of receiving messages was not 10000 as expected, this was caused by the difference in the starting time of the applications in both computers.

In figure 5.4 a comparison between signing and verifying for both messages is presented. The mean time to sign and to verify a message regardless of the size (53 or 67 bytes) it is $3.8504ms$ and $4.4157ms$, respectively.

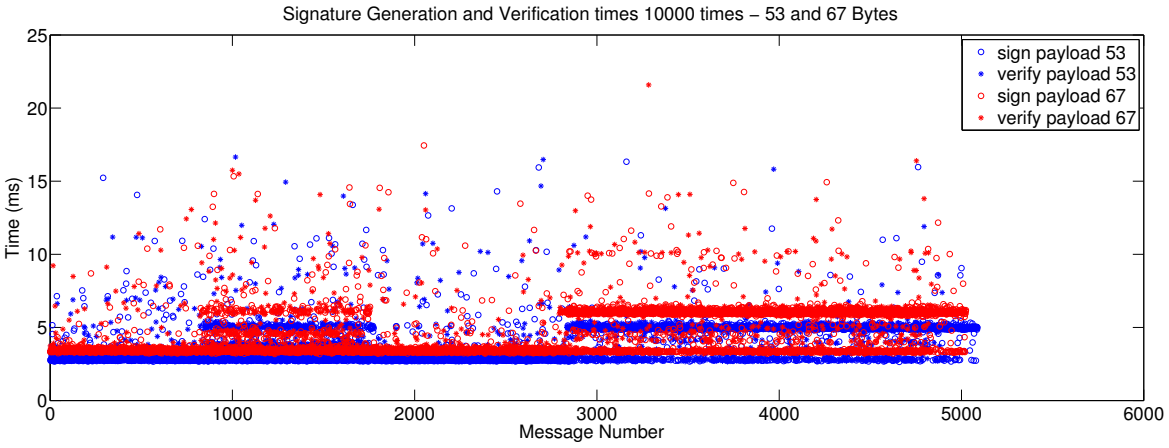


Figure 5.4: Execution times of security model with CAM message as payload.

The execution time, for either 53 or 67 byte messages, is very close because the red and blue colors are very intense meaning that their times are coincident. The average time for signing and verifying for either 53 and 67 byte messages is presented in table 5.4.

Message byte length	Signature (ms)	Verification (ms)
53	3.8684	4.4750
67	3.8318	4.4430

Table 5.4: Average time for signature and verification with CAM as payload.

Raspberry-Pi

Now it is time to analyse the timings on the Raspberry-Pi. For this test two Raspberry-Pi were communicating though Ethernet and sharing CAM messages at the frequency of 10Hz with its size changing between 53 and 67 bytes. Figure 5.5 shows that the times to perform a signature generation and verification are very high with the following mean times: signature generation: $21.7615ms$ and signature verification: $25.3628ms$, considering both the 53 and 67 byte payload of the CAM message.

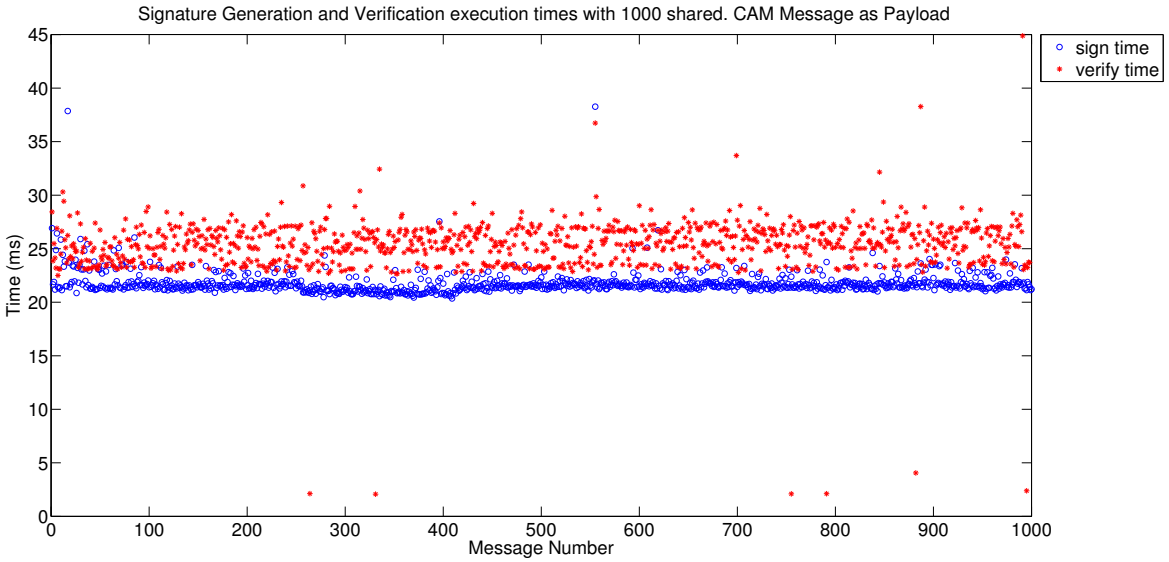


Figure 5.5: Signature generation and verification on Raspberry-pi.

These times were taken considering the whole system which is affected by the secure protocols and the gathering of the Global Positioning System (GPS) values. In the following table 5.5 there is a summary regarding the execution times.

Computer:	Raspberry-Pi
generation[ms]:	21.7615
Signature/s:	45
verification[ms]:	25.3628
Verification/s:	39
Maximum cars to verify [10 Hz]:	3.9

Table 5.5: Table with sum-up values for the whole system with Raspberry-Pi.

With this implementation, 4 is the maximum number of cars that it is possible to verify, broadcasting at a frequency of 10 Hz. The number of signature generation is fulfilled by the system because it is only required that each car signs 10 messages per second.

Chapter 6

Conclusions and Future Work

The main focus of this dissertation was to analyse the requirements and how much effort had to be done to provide the adequate security requirements for a vehicular network. The software proposed for the implementation was based on the IEEE 1609.2 TM/D17 Standard.

The conclusions taken from this work are the same as Papadimitratos in [31] and Rohde & Schwarz SIT GmbH project [25], meaning that a pure software implementation of security services is not feasible.

The constraints imposed by the on board units, which have a very low computation power, make the system very slow when security is added to the network. From the performance tests, considering the entire system architecture running on a laptop, the maximum number of signature verifications that the system is capable of performing is about 223 per second. These results mean that if cars are beaconing at 10Hz, the maximum number of cars in the neighbourhood possible to verify is ≈ 22, which is very common to happen in a congested road. These values were obtained using a laptop that has a higher performance compared with the Raspberry-Pi that was considered as the On Board Unit (OBU).

When using the Raspberry-Pi the performance goes down to 39 signature verifications per second, allowing the system to only verify ≈ 4 cars per second when beaconing at 10Hz. This is a very bad performance because in a congested highway the number of cars can be massive, requiring a system that must be capable of verifying several hundreds of cars at the same time.

From this results, it is understandable why there is the need to build a Hardware Security Model (HSM) to achieve the desired/needed performance. An Field-Programmable Gate Array (FPGA) implementation of the cryptography algorithms seems to be the best approach. OpenSSL libraries can be integrated with hardware FPGA making possible for the hard time consuming operations as the signature generation and verification to be performed purely in hardware. The HSM is proposed in the ETSI ITS Stack (please see figure 2.7) as an approach to achieve the desired performance.

Although the performance of the security services is not very good, the integration with the WAVE Short Message Protocol (WSMP) and Facilities Layer was a success, achieving a full-blown implementation that is ready to be integrated with the IT^{2}S Platform.

6.1 Future Work

As future work the study of other algorithms besides the ones referred in the IEEE 1609.2 TM/D17 Standard should be analysed. Some papers state that the use of Colbitz curves could be used instead of the NIST curves required by the security standard. A comparison of these algorithms in terms of performance could be analysed.

Also a hardware solution based on FPGA could be developed to perform some of the arithmetic operations required by the cryptographic algorithms. The hardware module can be integrated with the OpenSSL library allowing some hard operations to be performed on hardware.

Regarding the public key infrastructure, there should be an authority that could issue the WAVE Certificates with the specific fields defined in the IEE 1609.2 D17 Draft Standard. Also finding solutions for sharing certificates in an efficient way deserved an investment in further studies.

The IEEE 1609.2-2013 Standard released at the end of April 2013 has finally left the draft version, being the actual active standard to provide adequate security on Vehicular Communications (VC). It should be analysed because some important changes might have been done.

Security over vehicular communications is still an open issue which means there is yet a lot of research to be done in order to fully satisfy the security requirements of a vehicular communication network.

Bibliography

[1] WARDSAUTO. (2010) World Vehicle Population Tops 1 Billion Units. Accessed on June 2013. [Online]. Available: http://wardsauto.com/ar/world_vehicle_population_110815

[2] Federal Communications Commission (FCC). (2007) Dedicated Short Range Communications (DSRC) Service. Accessed on May 2013. [Online]. Available: http://wireless.fcc.gov/services/index.htm?job=service_home&id=dedicated_src

[3] B. S. Niels Ferguson and T. Kohno, *Cryptography Engineering*. John Wiley & Sons, 2010.

[4] Brisa, "HEADWAY - Connecting Vehicles and Highways," http://www.brisainovacao.pt/en/innovation/projects/headway.

[5] K. P. Hannes Hartenstein, *VANET: Vehicular Applications and Inter-Networking Technologies*, K. P. Hannes Hartenstein, Ed. WILEY, 2010.

[6] ARINC. Dedicated Short-Range Communications (DSRC). [Online]. Available: http://www.arinc.com/products/intel_trans_sys/dsrc.html

[7] Georgios Karagiannis, Onur Altintas, Eylem Ekici, Geert Heijenk, Boangoat Jarupan, Kenneth Lin, and Timothy Weil, "Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions," *IEEE Communications Surveys and Tutorials*, vol. 13, 2011.

[8] Li, Yunxin (Jeff), "An Overview of the DSRC/WAVE Technology." in *QSHINE*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, X. Zhang and D. Qiao, Eds., vol. 74. Springer, 2010, pp. 544–558. [Online]. Available: http://dblp.uni-trier.de/db/conf/qshine/qshine2010.html#Li10

[9] The Internet Engineering Task Force (IETF), "RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification."

[10] IEEE, "IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services," 2010.

[11] ETSI EN 302 665 V1.1.1, "ETSI EN 302 665 V1.1.1 Intelligent Transport Systems (ITS); Communications Architecture," 2010.

[12] CYCOM Cypher Research Laboratories, "A Brief History of Cryptography." 2006.

[13] Luringen. (2007) Scytale. Accessed on July 2013. [Online]. Available: http://commons.wikimedia.org/wiki/File:Skytale.png

[14] Mozilla Developer Network. (2005) Introduction to Public-Key Cryptography. [Online]. Available: https://developer.mozilla.org/en-US/docs/Introduction_to_Public-Key_Cryptography#A_Certificate_Identifies_Someone_or_Something

[15] Eli Biham and Adi Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, 1991.

[16] FIPS PUB 197, "Advanced Encryption Standard (AES)," 2001.

[17] S. V. Darrel Hankerson, Alfred Menezes, *Guide to Elliptic Curve Cryptography.* Pringer, 2004.

[18] Vasant Patel, Dr. Kris Gaj, "Key Sizes Selection in Cryptography and Security Comparison between ECC and RSA."

[19] R. Laboratories, *SA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1.* RSA Security Inc., 2000.

[20] S. L. Carlisle Adams, *Understanding Pki: Concepts, Standards, and Deployment Considerations.* Addison-Wesley Professional, November 16, 2002.

[21] IEEE P1609.2TM/D17, "Draft Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages," 2012.

[22] FIPS PUB 180-3, "Secure Hash Standard," 2008.

[23] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, and Tadayoshi Kohno, "Experimental Security Analysis of a Modern Automobile," *Department of Computer Science and Engineering University of Washington*, 2010.

[24] Stephen Checkoway, Damon McCoy, Brian Kantor, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," 2011.

[25] Rohde & Schwarz SIT GmbH, "Automotive Security: Cryptography for Car2X Communication," 2011.

[26] Kimmo Jarvinen, "Final Project Report: Cryptoprocessor for Elliptic Curve Digital Signature Algorithm (ECDSA)," 2007.

[27] Benjamin Glas, Oliver Sander, Vitali Stuckert, Klaus D. Müller-Glaser, and Jürgen Becker, "Prime Field ECDSA Signature Processing for Reconfigurable Embedded Systems," *International Journal of Reconfigurable Computing Volume 2011 (2011), Article ID 836460, 12 pages*, 2010.

[28] Panagiotis Papadimitratos,Levente Buttyan, Tamás Holczer, "Secure Vehicular Communication Systems: Design and Architecture," *IEEE Communications Magazine*, 2008.

[29] Escrypt. CycurV2X. Accessed on June 2013. [Online]. Available: https://www.escrypt.com/products/cycurv2x/overview/

[30] Ghassan Samara, Wafaa A.H. Al-Salihy, R. Sures, "Security Analysis of Vehicular Ad Hoc Networks (VANET)," *Second International Conference on Network Applications, Protocols and Services*, 2010.

[31] Frank Kargl, Panagiotis Papadimitratos, "Secure Vehicular Communication Systems: Implementation, Performance, and Research Challenges," *IEEE Communications Magazine*, 2008.

[32] FIPS PUB 186-3, "Digital Signature Standard (DSS)," 2009.

[33] Michael J. Donahoo and Kenneth L. Calvert, "TCP/IP Sockets in C: Practical Guide for Programmers."

[34] (2013, May) A GPS Service Deamon. Accessed on May 2013. [Online]. Available: http://catb.org/gpsd/index.html

[35] OpenSSL Documents. (2013) OpenSSL Crypto Docs. [Online]. Available: http://www.openssl.org/docs/crypto/

[36] Valgrind. Valgrind. Accessed on July 2013. [Online]. Available: http://valgrind.org/

[37] ETSI TS 102 637-2 V1.2.1, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service," 2011.

[38] ETSI TS 102 637-3 V1.1.1, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service," 2010.

[39] Arnaldo S. R. Oliveira and João Nuno Matos, "IT2S Board Description," *Instituto de Telecomunicações – Pólo de Aveiro*, March 2013.

[40] Bionic Buffalo Tech, "How Encryption and Digital Signatures Work," *Bionic Buffalo Tech Note 35*, 1999.

[41] Brian Wallace. (2012) DSA/ECDSA/SHA1 Benchmark. Accessed on July 2013. [Online]. Available: https://gist.github.com/bwall/3278083

[42] Dr. Michele Weigle, "Standards: WAVE / DSRC / 802.11p," 2010.

[43] ETSI TS 102 637-1 V1.1.1, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 1: Functional Requirements," 2010.

[44] H. Krishnan, "Vehicle Safety Communications (VSC) Project," February 15, 2006.

[45] IBM. (2013) IBM Systems Cryptographic Hardware Products. IBM. Accessed on May 2013. [Online]. Available: http://www-03.ibm.com/security/cryptocards/

[46] Kimmo Jarvinen, "Design and Implementation of a SHA-1 Hash Module on FPGAs," 2004.

[47] Marcus Bannerman, "Supercomputing on Graphics Cards," *Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)*.

[48] Miguel Morales-Sandoval and Claudia Fergrino-Uribe, "A Hardware Architecture for Elliptic Curve Cryptography and Lossless Data Compression," *IEEE*, 2005.

[49] Nisse. (2013) ECC Benchmark. Accessed on July 2013. [Online]. Available: http://git.lysator.liu.se/nettle/se-nettle-2013/blobs/ca37e2f0d88a288d5eecaacb8b27d6adc96f6139/benchmark/ecc-2013-02-27

[50] Roberto A. Uzcátegui, Guillermo Acosta-Marum, "WAVE: A Tutorial," *IEEE Communications Magazine*, 2009.

[51] J. W. . Sons, *Applied Cryptography - Protocols, Algorithms and Source Code in C*. BRUCE SCHNEIER, 1996.

[52] U.S Department of Transportation, "Vehicle Safety Communications – Applications (VSC-A)," 2011.

[53] B. Williams, *Intelligent Transport Systems Standards*, 2008.