Implementation of Security Services for Vehicular Communications

Daniel Duarte[1,2], Luis Silva[1,2(✉)], Bruno Fernandes[1], Muhammad Alam[1], and Joaquim Ferreira[1,3]

[1] Instituto de Telecomunicações, Aveiro, Portugal
brunofernandes@ua.pt, alam@av.it.pt
[2] DETI, Universidade de Aveiro, Aveiro, Portugal
daniel.duarte@ua.pt, luissilva@av.it.pt
[3] ESTGA, Universidade de Aveiro, Aveiro, Portugal
jjcf@ua.pt

Abstract. Over the last years, there has been a considerable development in the field of vehicular communications (VC) so as to satisfy the requirements of Intelligent Transportation Systems (ITS). Standards such as IEEE 802.11p and ETSI ITS-G5 enable the so called Vehicular Ad-Hoc Networks (VANETs). Vehicles can exploit VANETs to exchange information, such as alerts and awareness information, so as to improve road safety. However, due to the expected popularity of ITS, VANETs could be prone to attacks by malicious sources. To prevent this, security standards, such as IEEE 1609.2, are being developed for ITS. In this work, an implementation of the required cryptographic algorithms and protocols for the transmission of secure messages according to the IEEE 1609.2 standard is presented. The implemented security protocols are then integrated into an existing WAVE-based system and tested in a real scenario to evaluate the performance impact on safety-related communications, in particular, the overhead that is caused by the process to sign/verify a digital message.

Keywords: Vehicular Communications · Intelligent Transportation Systems · Security · IEEE 1609.2

1 Introduction

During the past decades, the volume and density of road traffic has increased significantly, specially in developing countries such as India and Brazil. According to [1], in 2010 the number of vehicles in the world has reached to 1.015 billion, with an approximate ratio of 1:7 cars per person. This significant growth lead to an increase in number of accidents and traffic injuries, with negative impacts on the economy and in the quality of people's lives [2]. In order to tackle this problem new systems, commonly known as Intelligent Transportation Systems (ITS), are being developed. In ITS, Dedicated Short Range Communications (DSRC), on a dedicated spectrum in the 5.9 GHz band, are employed to enable wireless

J. Ferreira and M. Alam (Eds.): Future 5V 2016, LNICST 185, pp. 79–90, 2017.
DOI: 10.1007/978-3-319-51207-5_8

communication between vehicles and infrastructure. New applications can then be developed to exploit vehicle to vehicle (V2V) [8] and vehicle to infrastructure (V2I) [9] communications to improve road traffic's safety and efficiency. To this end, a group of new standards that specifies and standardizes several aspects of the aforementioned communications (e.g. physical and medium access control layers, data structures, security, etc.) has been defined. The most well known standards are the Wireless Access in Vehicular Environments (WAVE) in the USA and ETSI ITS-G5 in Europe.

Due to the high popularity and scale that ITS can attain, communications in vehicular networks, more specifically in Vehicular Ad-Hoc Networks (VANETs), are expected to be prone to security threats since these could be exploited by people with malicious intents (e.g. redirect traffic flow and spread false information etc.). Eavesdropping, message manipulation and replay attacks are examples of attacks that may target a VANET. Hence, an architecture, a set of interfaces and services that enable secure V2V and V2I wireless communications are also included in WAVE and ETSI ITS-G5 group of standards [3].

Security objectives and solutions are well defined for computer-based architectures in general but for vehicular environments the approach needs to be different due to its distinct properties and requirements [5]. For example, robustness and time constraints in VC are demanding. Vital functions for driving and/or alerts sent by other vehicles must be correctly processed in real-time; delays or errors might lead to vehicle malfunctions, bad driving decisions, or other occurrences that could cause physical damages and injuries. The small embedded computers found in vehicles may not have the necessary memory and performance for cryptographic operations without affecting the aforementioned functions. Moreover, since a car typically has a life-time of at least 10 years, an upgrade of the computational resources may not be possible and thus, it is important to assure all security requirements for cars' life time-frame. This work aims to assess the requirements of such security mechanisms for vehicular environments as well as to evaluate the performance impact that such mechanisms may induce in communications. To that end, a software implementation of the IEEE P1609.2 [5], the current security standard embedded in WAVE, will be presented and evaluated in a real world scenario.

This paper is structured as follows. Section 2 provides a brief overview of main features of the IEEE 1609.2 standard. Section 3 presents the software implementation of IEEE 1609.2 algorithms. Section 4 presents the integration of the implemented software algorithms into an existing WAVE-based framework. In Sect. 5 experimental scenarios to evaluate the performance of the implementation are described and the results presented. The paper concludes with the discussion of the obtained results in Sect. 6.

2 IEEE 1609.2 Brief Overview

Our implementation is based on the IEEE security standard 1609.2 D17 [6]. 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 key, the ECIES a 256-bit and the AES a 128-bit key implementation. 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. Most of the topics are still open, such as how certificates should be shared among elements on the road and which 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.

3 IEEE 1609.2 Implementation

3.1 System Architecture

In this section, a software implementation of the security services is proposed. C programming language was chosen due to its performance and available open-source libraries. OpenSSL [4] was picked as the auxiliary library to implement the security services and the cryptography engine since it is a free and open-source c/c++ library tailored for cryptographic operations and algorithms.

The implementation architecture was divided into 3 main modules: one to handle all the secure protocols, another responsible for the cryptographic algorithms and finally, 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. In addition 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 [7] which is illustrated in Fig. 1. The engine which performs all the cryptographic algorithms was implemented with the OpenSSL library and it is capable of performing Elliptic Curve Digital Signature Algorithm (ECDSA) 224 and 256, ECIES, Hash algorithms, certificate generation and verification.

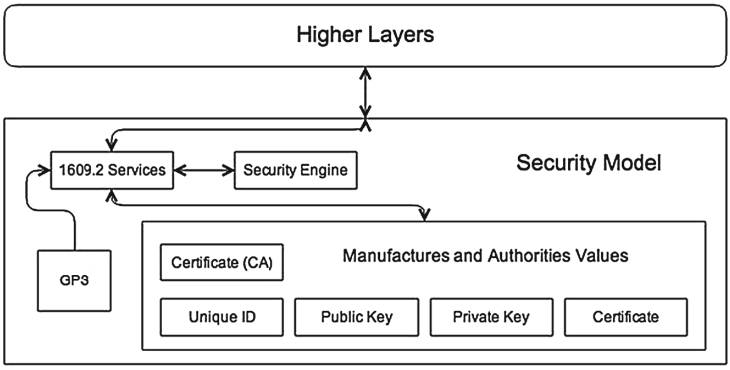

Fig. 1. Security model

Software Resources: The following set of software and libraries were used in the implementation.

Hash Table: A hash table is an abstraction of a common array in which contents can be accessed using keys instead of indexes. These tables allow unordered data to be searched and accessed very quickly.

Serialization: Serialization is the process of transforming data structures into a buffer of data, suitable for storage or transmission (the reverse process is also possible). This serialization mechanism is used to process structures and packets defined by the 1609.2 standard.

Sockets: When the connection between two programs or two machines is needed, the use of a mechanism called sockets can be used. Sockets will be used to interconnect certain software processes as well as the security module with the existing WAVE-based system.

GPSd: The GPSd is a daemon for GPS devices which interacts with various GPS brand devices through Universal Serial Bus (USB) serial interface enabling users to easily interact with different GPS devices and obtain GPS information. The GPSd is used in this implementation in order to get the time stamp and location to be inserted in the 1609.2 secure packet.

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. We assumed to have a fully Public-Key Infrastructure (PKI) deployed, thus, algorithms to share certificates and manage certificate-revocations were not considered. 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. Thus, in this implementation, 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.

Certificate hashes are included in the transmitted messages so as to avoid including the entire certificate. 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 [6]. Each car also contains a table of hashes, identifying all the stored certificates and its corresponding hash.

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. 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. Then the system awaits the reception of a message. As soon as it is received, its contents are analysed and a process to verify or sign the message is performed. The message is then returned to the facilities layer and the system stays again waiting for new messages. The process to create the signature generation or signature verification of a message is detailed in Fig. 2. There are two important functions defined by the IEEE 1609.2 Standard [6],

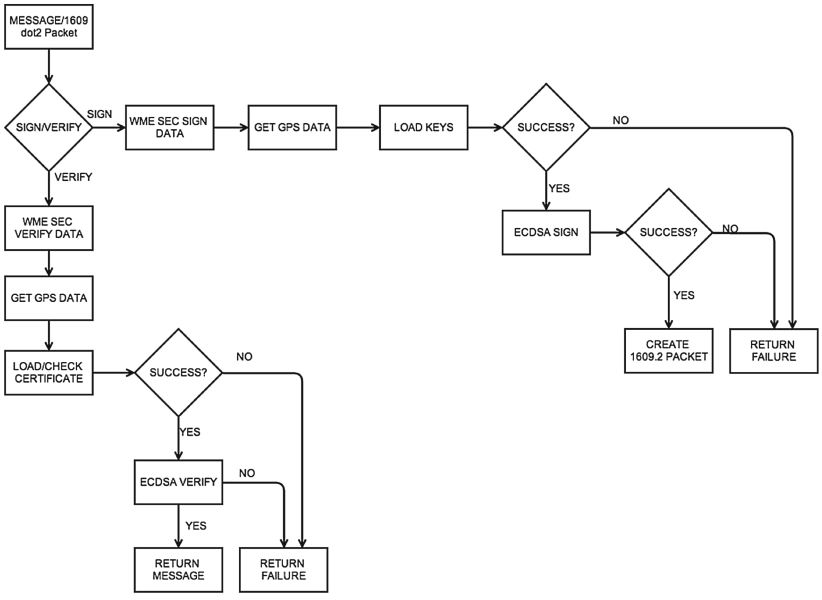

Fig. 2. Messages' signature generation and verification process

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. These functions access the GPS data and retrieve the keys.

3.2 ECDSA Implementation

The ECDSA algorithm was implemented in C programming language using the OpenSSL library.

Open-SSL API: In this section, the OpenSSL library which performs the required cryptographic algorithm with the specific curves is explained.

Key Generation: The key generation process is not ECDSA specific and it is generated in the following way [4]:

Algorithm 1. ECDSA Key generation

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

The NID value that is assigned in the function EC_GROUP 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 [4]. 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:

- eckey→ version = 1;
- eckey → group=NULL;
- eckey → pub_key=NULL;
- eckey → priv_key=NULL;
- eckey → enc_flag=0;
- eckey → references=1;
- eckey → method_data=NULL;
- eckey → conv_form=POINT _CONVERSION_UNCOMPRESSED;

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:

Algorithm 2. ECDSA Signature Generation

```
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, with twice the size of the key length used in the ECDSA algorithm.

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

Algorithm 3. ECDSA Signature Verification

```
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 function determines if the verification was a success (return value = 1), failure (return value = 0) or an error occurred (return value = −1).

Certificates: The IEEE 1609.2 Standard defines the WAVE-Certificates which are a special type of certificates for Vehicular Communications. As PKI was not implemented, the standard certificates generated using OpenSSL were used. The used certificates are encoded in a specific format (X.509) and 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 certificates had approximately 956-bytes which are too big when compared to the expected WAVE Certificate size of about 120-bytes [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.

3.3 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 interface 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. Secured data is sent over the network within a secure packet structure defined by IEEE 1609.2. Its structure is composed of multiple sub-structures as it is shown in Fig. 3. Figure 4 illustrates the implemented protocols for the signature generation and verification.

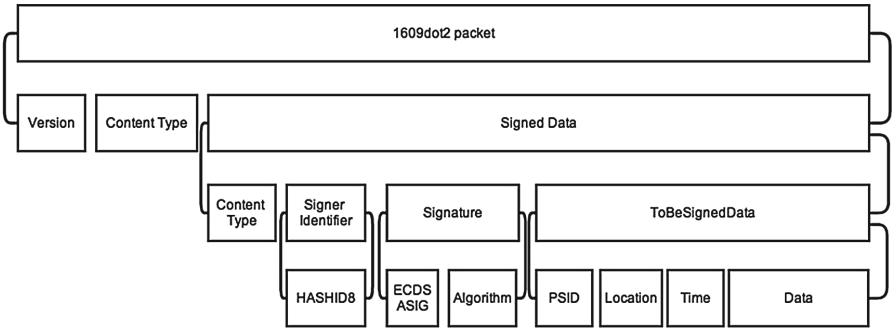

Fig. 3. 1609.2 secure packet struct

4 Integration with WSMP and Facilities Layer

After the security services have been developed there was the need to integrate this work with other applications such as the WAVE Short Message Protocol (WSMP) and the facilities layer in order to have a full system working. Sockets were used to interconnect the different applications and create an architecture capable of generating messages, secure them and communicate through the Dedicated Short Range Communications (DSRC) platform IT2S developed in the IT (Telecommunications Institute) in the scope of the FP7 project ICSI.

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 1609.2 packet, it is sent back to the facilities module which forwards its content to the WSMP. The WSMP generates the WSM packet and puts in its data field the 1609.2 packet received from the facilities module. This packet is then transmitted over the DSRC platform.

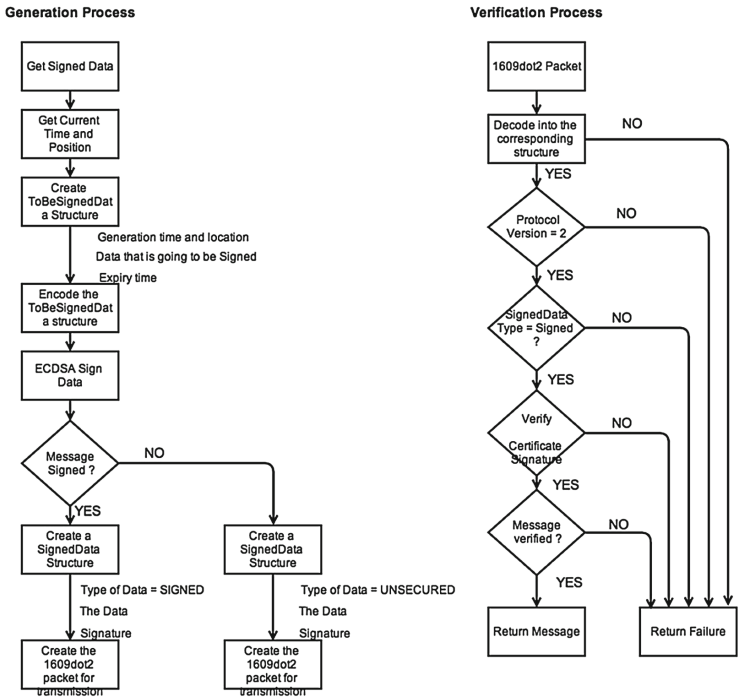

Fig. 4. Signature generation and verification protocol

Table 1. Flow detection and mitigation time

Specification	Personal laptop	Raspberry-Pi
CPU	Intl. Pentium 1.46 GHz	700 MHz Low Power
Instruction-Set	32-bits	32-bits
Memory	3 GB	512 MB SDRAM
OS	Ubuntu 12.04	Arch-Linux

5 Experimental Evaluation

This section presents a experimental performance analysis of the implemented architecture. The main focus of these experiments was to benchmark the overall system that provides authenticity with ECDSA. In the 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. Table 1 provides hardware and software comparison between the two used machines. The following set of experiments were carried out:

- 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 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. 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. 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 Table 2 summary of the mean execution time for both key algorithms and computers is presented.

Table 2. Experiment results for ECDSA 224 and 256

Computer	Algorithm	Sign [ms]	Signature/s	Verify [ms]	Verifications/s
Laptop	ECDSA P256	2.339	411	2.8676	349
Laptop	ECDSA P224	1.8958	527	2.222	450
Raspberry-Pi	ECDSA P256	11.3833	88	13.3581	75
Raspberry-Pi	ECDSA P224	8.7552	114	10.2238	98

It is important to mention that all these values only represent timings of the OpenSSL signature generation and verification functions. For the OpenSSL functions to sign and verify, an ECDSA signature or verification requires that a key-pair must be previously generated or loaded and a hash function must be applied to the payload. These values do not represent the overall system execution times but they were taken to benchmark the OpenSSL library.

5.2 Integration of CAM, WSMP and Security

Here we present 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 payload, the time for the whole process, the signature generation, the signature verification, loading the certificates and the hashing of data. We used a real testbed having all the defined modules integrated and transmitting the CAM messages at the rate of 10 Hz. Ten thousand

messages were sent at this rate to analyse the system performance. Two different size of messages were generated, one with 53 bytes and another one with 67 bytes. In Fig. 5, 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) is 3:8504 ms and 4:4157 ms respectively. Figure 5 shows that the times to perform a signature generation and verification are very high with the following mean times: signature generation: 21:7615 ms and signature verification: 25:3628 ms, considering both the 53 and 67 byte payload of the CAM message (Fig. 6).

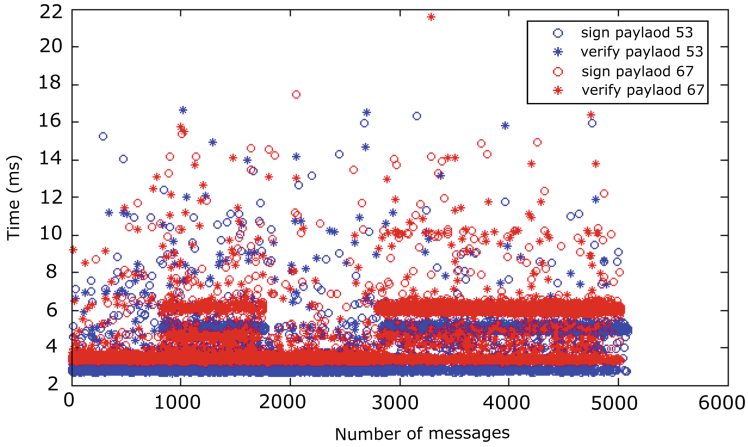

Fig. 5. CAM timings obtained on laptop

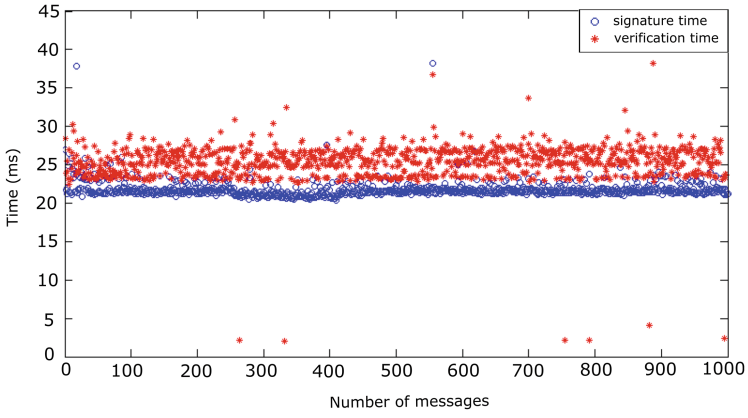

Fig. 6. CAM timings obtained on Raspberry Pi

6 Conclusions

The main focus of this work was to analyse the requirements and how much effort has to be applied to provide the adequate security requirements for a vehicular network. The proposed software implementation was based on the IEEE 1609.2 TM/D17 standard and the entire system was evaluated in a real world scenario. Performance tests show that the system is capable of performing 223 and 39 signature verifications per second when running on a conventional laptop or Raspberry-Pi respectively. Since in WAVE, cars are typically beaconing at 10 Hz, the maximum number of cars in the neighbourhood possible to verify are 22 using laptop and 4 using RaspBerry-Pi respectively. However in real world scenarios, specially in highway and congestion scenarios, the number of vehicles in a given area can reach to hundreds. Thus, it can be concluded that the performance of a pure software implementation of security services is insufficient for real world use-cases. As future work, the study of other algorithms besides the ones referred in the IEEE 1609.2 TM/D17 Standard should 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.

Acknowledgment. This work was funded by the European Union's Seventh Framework Programme (FP7) under grant agreement no. 3176711.

References

1. Sousanis, J.: WardsAuto. World Vehicle Population Tops 1 Billion Units, August 2011. http://wardsauto.com/news-analysis/world-vehicle-populationtops-1-billion-units
2. Alam, M., Ferreira, J., Fonseca, J.: Introduction to intelligent transportation systems. In: Alam, M., Ferreira, J., Fonseca, J. (eds.) Intelligent Transportation Systems. SSDC, vol. 52, pp. 1–17. Springer, Heidelberg (2016). doi:10.1007/978-3-319-28183-4_1
3. Alam, M., Fernandes, B., Silva, L., Khan, A., Ferreira, J.: Implementation and analysis of traffic safety protocols based on ETSI Standard. In: Proceedings of the 2015 IEEE Vehicular Networking Conference (VNC), pp. 143–150, Kyoto (2015)
4. Hudson, T., Young, E.: OpenSSL - Cryptography and SSL/TLS Toolkit. https://www.openssl.org/. Accessed 30 June 2016
5. Hartenstein, H., Laberteaux, K.: VANET Vehicular Applications Inter-Networking Technologies. Wiley, Hoboken (2010). ISBN: 978-0-470-74056-95
6. IEEE Standard for Wireless Access in Vehicular Environments Security Services for Applications and Management Messages. doi:10.1109/ieeestd.2013.6509896
7. Papadimitratos, P., et al.: Secure vehicular communication systems: design and architecture. IEEE Commun. Mag. **46**(11), 100–109 (2008). doi:10.1109/mcom.2008.4689252
8. Alam, M., Sher, M., Afaq Husain, S.: VANETs mobility model entities and its impact. In: 2008 4th International Conference on Emerging Technologies, October 2008
9. Alam, M., Fernandes, B., Silva, L., Khan, A., Ferreira, J.: Implementation and analysis of traffic safety protocols based on ETSI Standard. In: Proceedings of the 2015 IEEE Vehicular Networking Conference (VNC), December 2015