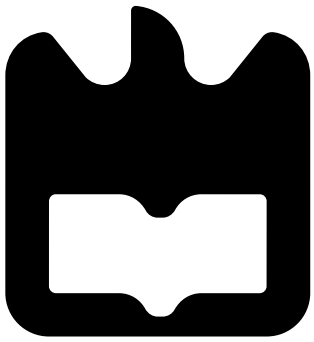
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2015

Gonçalo Miguel
Vieira Gomes

# Encaminhamento e Segurança em Redes Veiculares

# Forwarding and Security in Vehicular Networks



Gonçalo Miguel
Vieira Gomes

# Encaminhamento e Segurança em Redes Veiculares

# Forwarding and Security in Vehicular Networks

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 André Zúquete, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

**o júri / the jury**

presidente / president

**Professor Doutor Tomás António Mendes Oliveira e Silva**
Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

**Professor Doutor Pedro Miguel Alves Brandão**
Professor Auxiliar do Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto (arguente)

**Professor Doutor André Zúquete**
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática de Aveiro (Orientador)

**agradecimentos**

Em primeiro lugar, gostaria de agradecer à minha família pelo apoio fornecido ao longo dos anos, pois sem eles não seria possível obter formação e também aos meus amigos que me apoiaram ao longo do percurso académico.

Um agradecimento aos elementos do NAP que me ajudaram a perceber alguns detalhes de implementação que de outra maneira poderiam ter demorado bastante mais tempo a resolver.

Gostaria de agradecer ao Professor André Zúquete e à Professora Susana Sargento, que me orientaram durante toda a duração da tese e com os quais aprendi e evoluí bastante.

Gostaria também de agradecer à Veniam, que forneceu acesso à rede de investigação no Porto de Leixões, com o objetivo de testar a solução desenvolvida em ambiente real.

**Resumo**

O crescimento da investigação em redes veiculares provocou o aumento da interação nestes ambientes muito dinâmicos no mercado. As arquiteturas desenvolvidas não se focam, no entanto, na segurança. Estratégias comuns de segurança para a Internet, requerem sessões baseadas no IP. Como os endereços dos nós numa rede veicular, e a sua localização e caminhos até à Internet, são muito dinâmicos, as soluções já desenvolvidas para outro tipo de redes iriam requerer renegociações que teriam um grande impacto no desempenho destes ambientes.

O objetivo desta dissertação será, portanto, desenvolver e testar um protocolo de segurança implementado na camada 3 para redes veiculares, que seja escalável e leve, em que os nós da rede conseguirão estabelecer associações de segurança de longa duração com a *Home Network*, evitando renegociações devidas à falta de conectividade, e reduzir o *overhead* devido ao empilhamento protocolar. Este protocolo permite ter segurança independentemente da posição dos nós (os veículos), do seu endereçamento e do caminho estabelecido para o acesso à Internet, permitindo assim mobilidade dos veículos e das sessões ativas de forma transparente sem falhas na comunicação.

**Abstract**

The growing research in vehicular network solutions provided the rise of interaction in these highly dynamic environments in the market. The developed architectures do not usually focus, however, in security aspects. Common security strategies designed for the Internet require IP. Since nodes' addresses in a vehicular network are too dynamic, such solutions would require cumbersome negotiations, which would make them unsuitable to these environments.

The objective of this dissertation is to develop, and test a scalable, lightweight, layer 3 security protocol for vehicular networks, in which nodes of the network are able to set up long-term security associations with a Home Network, avoiding session renegotiations due to lack of connectivity and reduce the protocol stacking. This protocol allows to provide security independent of the nodes (vehicles) position, of its addressing and of the established path to access the Internet, allowing the mobility of vehicles and of its active sessions seamlessly without communication failures.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **AAD** | Additional Authenticated Data |
| **AEAD** | Authenticated Encryption with Associated Data |
| **AES** | Advanced Encryption Standard |
| **AH** | Authentication Header |
| **ARP** | Address Resolution Protocol |
| **CBC** | Cipher Block Chaining |
| **CFB** | Cipher Feedback |
| **CHAP** | Challenge Handshake Authentication Protocol |
| **CMS** | Cryptographic Message Syntax |
| **CTR** | Counter |
| **DSRC** | Dedicated short-range communications |
| **DHP** | Diffie-Hellman Problem |
| **DH** | Diffie-Hellman |
| **DLP** | Discrete Logarithm Problem |
| **DoS** | Denial of Service |
| **ECB** | Electronic Code Book |
| **EDA** | Event-driven architecture |

| | |
|---|---|
| **EID** | Endpoint Identifier |
| **ESP** | Encapsulating Security Payload |
| **GAF** | Geographic Adaptive Fidelity |
| **GCM** | Galois-Counter Mode |
| **GPS** | Global Positioning System |
| **GRE** | Generic Routing Encapsulation |
| **HMAC** | Hash Message Authentication Code |
| **HN** | Home Network |
| **ICV** | Integrity Check Value |
| **IKE** | Internet Key Exchange |
| **IPSec** | Internet Protocol Security |
| **IPSP** | IP Security Protocol |
| **IPX** | Internetwork Packet Exchange |
| **ISAKMP** | Internet Security Association and Key Management Protocol |
| **ITS** | Intelligent Transport System |
| **IV** | Initialization Vector |
| **KINK** | Kerberized Internet Negotiation of Keys |
| **MAC** | Message Authentication Code |
| **MANET** | Mobile Ad Hoc Network |
| **MitM** | Man-in-the-Middle |
| **MIP** | Mobile IP |
| **MOBIKE** | IKEv2 Mobility and Multihoming Protocol |
| **NAS** | Network Access Server |

| | |
|---|---|
| **NAT** | Network Address Translation |
| **NetBIOS** | Network Basic Input/Output System |
| **NIC** | Network Interface Card |
| **NIST** | National Institute of Standards and Technology |
| **OBU** | On Board Unit |
| **OFB** | Output Feedback |
| **PAP** | Point-to-Point Authentication Protocol |
| **PFS** | Perfect Forward Secrecy |
| **PKCS** | Public-Key Cryptography Standards |
| **PKI** | Public-Key Infrastructure |
| **PPP** | Point-to-Point Protocol |
| **PPTP** | Point-to-Point Tunnelling Protocol |
| **RAM** | Random Access Memory |
| **RED** | Random early detection |
| **RSU** | Road Side Unit |
| **SA** | Security Association |
| **SHA** | Secure Hash Algorithm |
| **SPI** | Security Parameter Index |
| **SSL** | Secure Sockets Layer |
| **TCP** | Transmission Control Protocol |
| **UDP** | User Datagram Protocol |
| **VANET** | Vehicular Ad Hoc Network |
| **VEID** | Vehicle Endpoint Identifier |

| | |
|---|---|
| **VPNC** | Virtual Private Network Consortium |
| **VPN** | Virtual Private Network |
| **WLAN** | Wireless Local Area Network |
| **WME** | Wave Management Entity |
| **WPA2** | Wi-Fi Protected Access II |

# Chapter 1

# Introduction

The concept of Vehicular Ad Hoc Networks (VANETs) is now a reality. One of the main VANETs deployment was the result of a partnership between, among other stakeholders, University of Aveiro and Porto, Instituto de Telecomunicações and Veniam. This network comprises the mobile devices deployed in vehicles as mobile access points, and the fixed infrastructure which establishes a connection between the mobile devices and the internal network.

## 1.1 Motivation

Communication is a necessity integrated in the current society. VANETs bring many useful services, such as traffic control, content distribution, real-time content access. In comparison with cellular infrastructures, VANETs may be composed by more nodes than cellular; however, the access technology may be cheaper than cellular, the vehicle to vehicle communicatons are made easily, and nodes are powered by car batteries, which recharge while the car is moving. This results in a network which is composed by nodes connected in a meshed fashion, and operators can take advantage of this meshed capabilities to extend the range and provide services to users by routing traffic to gateways cooperatively.

However, it is important that, in such a dynamic networks, unauthorized third parties, which may enter the network, do not compromise the integrity and performance of legitimate nodes. Also, privacy must not be compromised in a network where the channel is shared. Common solutions are bound to IP, and depend on security mechanisms which make such solutions inappropriate networks with constant loss of connectivity.

Furthermore, existent VPN solutions could be integrated alongside with Mobile IP, but

the overhead caused by stacking these two solutions, would cause strain on the network. Because of that, there is a need to design and implement a solution able to maintain security on these networks without compromising performance, and enabling fast mobility seamless for the network and the running communications.

Home Network (HN) is a network created in the core of the provider, which may supply several services to mobile access points. We want to provide a secure connection to this network through a mesh network composed by vehicles and cabled infrastructure.

The elements of this mesh network should cooperate between them to establish valid connections to the HN but, at the same time, the traffic generated from each element should remain confidential between itself and the internal provider network.

In order to achieve this, the security was subdivided into two segments. The mobile router deployed in a vehicle authenticates in the HN through the vehicular network, and then, user authentication uses common solutions to authenticate the users inside the vehicles.

## 1.2 Problem

Until today, to the best of our knowledge, all VPNs are IP based. Parameters are bound to IP which cause frequent renegotiation when mobility occurs, and several protocols are stacked. Moreover, there is not a solution for secure communications in VANETs seamless to the mobility of the vehicles.

Since mobility occurs frequently, those protocols, which were designed to operate with sporadic mobility would impose frequent downtimes, and a user accessing the service would not be able to use the network.

## 1.3 Objectives

The purpose of this dissertation is to establish a secure connection between the user connected to a vehicle through its mobile access point, the On Board Unit (OBU), to the HN where its services are deployed. The solution should behave like a VPN and should attain the following advantages:

- Remove overhead of protocol stacking, providing a light protocol for the dynamic nature of the vehicular networks.

- Replace the usual internet mechanism which binds session parameters to IP, by a mechanism which binds vehicle identifiers to session parameters. In opposition to IP, the vehicle identifier is controlled in the vehicular network, and maintained constant at all times. Because of that, keys do not need to be renegotiated constantly, in a network where connection is intermittent.

Also, the security solution should separate the routing strategy from the security approach, which will allow the routing algorithm to be changed without compromising the security.

## 1.4 Contributions

In order to gain insight about the architecture's requirements, security primitives and protocols were studied.

The main focus of the dissertation was to implement a solution which allowed to deploy a lightweight security architecture in VANETs. To that end, a routing mechanism based on fixed identifiers and an end-to-end security protocol were implemented.

This solution was deployed and tested not only in the laboratory, but also in a real environment of vehicles and road side units connected to a HN. The concept and its results were also presented in a scientific paper which was published in the 10th Conference on Telecommunications, Conftele 2015. A final journal paper is in preparation and will be submitted in the end of 2015.

## 1.5 Document Organization

This document is organized as follows.

In chapter 2, it is given a general overview about vehicular networks, its elements, and architecture. It is also presented the routing strategies applied in these networks, because they differ from the traditional solutions. Since this dissertation will focus on end-to-end security, the security primitives, protocols and concepts used in the document will also be explained. Also, existent solutions are presented, and it is discussed why they do not match the requirements for these networks.

In chapter 3, a high-level description of the proposed architecture is stated, and explained how that solution meets the requirements.

In chapter 4, we focus on describing how the solution was designed, which libraries are used and why, and a description of the structures implemented and used to hold the data.

In chapter 5, the solution is evaluated, both in terms of network performance and software performance.

In chapter 6, we expose the goals which have been met, and what needs to be done in order to further evolve the work of this dissertation.

# Chapter 2

# State of the art

## 2.1 Introduction

VANETs have different requirements than fixed networks, with that in mind, the nodes which compose the network also have different requirements than normal network devices. In section 2.2 it is described the core elements of this networks.

In section 2.3, the architecture used in these networks is described. These architectures describe how communication between the nodes is established.

In VANETs, the usual network routing strategies do not work, because of the implicit mobility in these networks. Furthermore, there is extensive research in suitable protocols. Thus, in section 2.4, several types of routing strategies which are commonly used in VANETs are described.

Since research has been, usually, focused in routing, and mobility, some concerns about these environments are presented in section 2.5. It is also described the respective implications on the network and user experience.

After exposing the concerns in such environments, solutions which solve them, need to be presented. In section 2.6, it is described the security requirements which a solution should achieve in these networks.

In section 2.7, it is explained the security concepts used through this document, the alternatives, and each one of their advantages and disadvantages. In this section, it is explored the primitives, such as hash functions, symmetric and asymmetric key algorithms, key exchange protocols. Then, it is given a perspective of the current VPN solutions used in the Internet, and why they are not used in this scope.

In section 2.8, it is described already existent architectures proposed in these environ-

Figure 2.1: Hardware used by the RSUs and OBUs

ments. These security solutions are, however, mostly used for different purposes than this dissertation. Because of that, this section focus on relating the purpose of the solution, with the security primitives used in it.

Section 2.9, presents conclusion about the existing solutions and points what can be improved in them, regarding the primitives presented.

## 2.2 Equipment

A VANET architecture, is composed by two types of nodes.

- The mobile nodes, OBUs, are nodes which are deployed in moving vehicles. They make use of the energy supplied by the vehicle's battery and communicate using other OBUs or other fixed access points.
- The fixed infrastructure, Road Side Units (RSUs), are nodes which are deployed close to the road, where the OBUs are expected to be seen. They allow communication of OBUs with the exterior network, or even to send data across vehicles. Because they are fixed, they may use high-speed, fixed infrastructure between themselves, and communicate with mobile nodes through 802.11p.

## 2.3 Architecture

A VANET encompasses various types of communications, as shown in Figure 2.2.

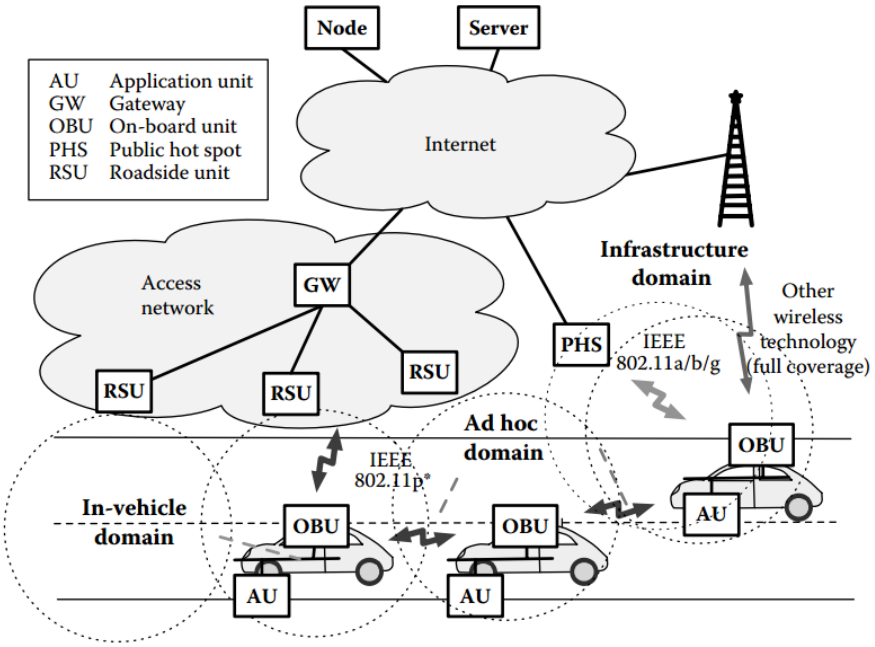



Figure 2.2: Architecture of a VANET network [38]

- V2V: Vehicle to Vehicle is the most dynamic type of communication, and also the most common one in this type of environments. Nodes establish temporary routes using different types of algorithms, and route data between themselves using ad-hoc/sensor routing strategies conceived for establishing connection between nodes in dynamic, and energy constrained networks.

- V2I/I2V: Vehicle to Infrastructure and vice-versa is the type of communication where a node can use the fixed infrastructure to communicate with external networks, and also allow the outside network to communicate with nodes in the VANET.

By using all the VANET nodes in cooperation, one can extend the maximum range of the network several times in comparison to normal centralized wireless technology, and provide services to users in vehicles in a dynamic and large scale.

These networks provide services, such as, traffic monitoring and control, accident prevention, and others which can be used to improve road safety.

They are also used, to provide access to entertainment services, through a network where a provider might deploy applications and gateways to external networks.

The channels allocated for VANETs, described by Dedicated short-range communications (DSRC) are shown in Figure 2.3. This set of standards provides seven 10 MHz

channels in the 5.9 GHz. One of them should be used as a control channel and the remaining six are used for service between nodes.

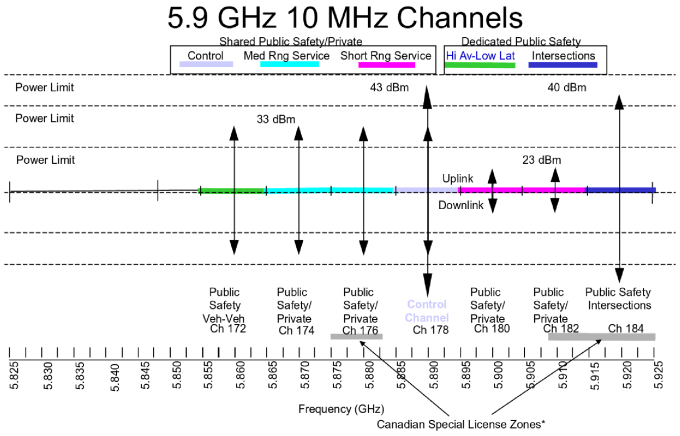



Figure 2.3: Channel allocation for DSRC [7, 25]

In these channels the stack used to communicate can be seen in Figure 2.4. This stack features the IEEE 1609.x which is used to control the resources between nodes, the security mechanisms and the WAVE' network layer.

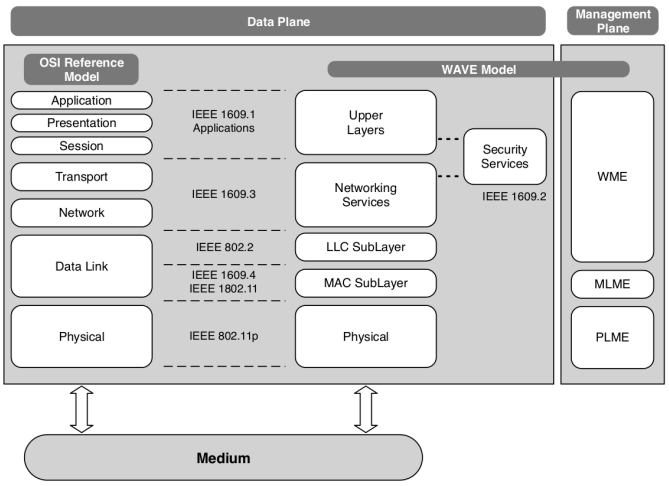



Figure 2.4: WAVE Protocol [6]

This technology also features a fast association which allows for very fast handovers as shown in Figure 2.5. The features missing in the fast association shown in the vehicular technology are usually passed to the upper layers, which might differentiate security approaches based on the service.

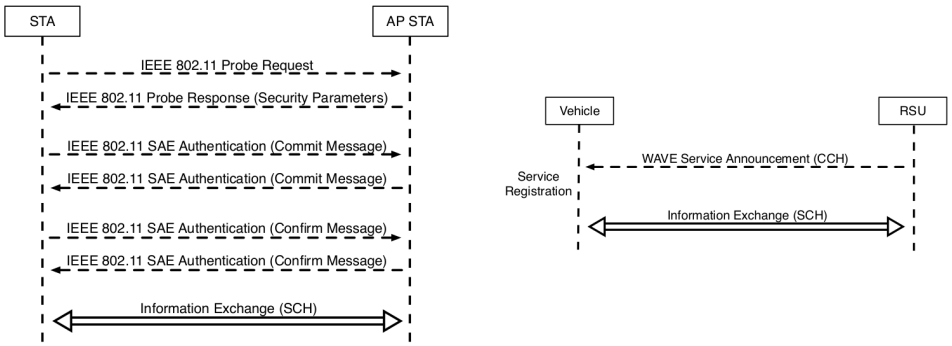



Figure 2.5: Comparison of the association processes between usual Wi-Fi (on the left) and IEEE 802.11p (on the right) [1]

## 2.4 Routing strategies

Since vehicles are in constant movement, routing strategies used in fixed networks cannot be applied, because they do not consider mobility. Therefore, several routing strategies are developed to be used in these dynamic environments. Considering that VANETs are a subclass of Mobile Ad Hoc Networks (MANETs), most the typical routing strategies implemented in both ad-hoc and sensor networks, are suitable to be applied here.

In the following subsections typical sets of routing strategies applied in these networks are presented.

### 2.4.1 Ad-hoc Networks Routing

In order to establish non-ambiguous routes, each node needs to have a unique address in the ad-hoc network.

The routing strategies should meet certain requirements, among them are the routing acquisition and reconfiguration delays, and also the network overhead, scalability, security and privacy.

The first question in ad-hoc routing is “Who determines routes?”. This question may be answered in two ways, each one with its advantages and disadvantages.

In source routing, the whole path is described in the packet. The intermediate nodes forward the packet to the next hop in the list. For this, the source must have a lot of storage, because it needs at least, all the best paths which describe the route for everyone. But it does not need to have storage in the intermediate nodes. In destination routing, the

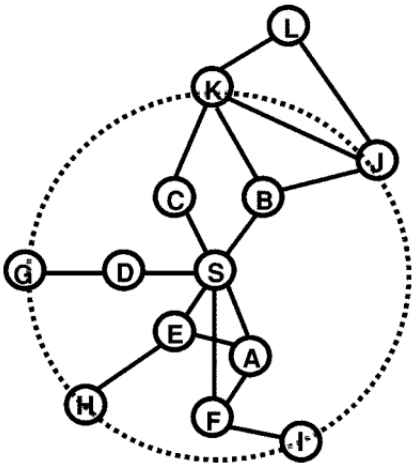



Figure 2.6: ZRP routing zone the node S with radius 2 [15]

source only specifies the destination in the packet. The source must hold, in memory, the next-hop for each node in the network.

These protocols might be also separated in other sets of protocols. The proactive protocols maintain the routes with every host at all times, and are based on periodic updates. The reactive protocols only determine route, on a needed basis. Whilst the first set always consumes a certain bandwidth, to keep routes updated, the second one employs flooding to search the whole network for one node, and because of that, reactive routing has bursty traffic.

Some protocols, such as the Zone Routing Protocol (ZRP) are hybrid, because they have separate components which have characteristics of both, proactive and reactive protocols. The ZRP protocol limits the proactive procedure to the node' local neighbourhood minimizing the bandwidth waste of proactive protocols. To search in between zones, a reactive scheme is used. In Figure 2.6, it is shown the radius of the $S$ node in 2 hops.

Other different class of algorithms is the hierarchical routing protocols, such as, the Fisheye State Routing (FSR). As seen on Figure 2.7, in order to diminish the bandwidth used, the packet is sent to neighbours and a full topology map is kept at each node.

The VANET subclass, increases the number of routing protocols existent, as Global Positioning System (GPS) is available at most times. Examples of such types of routing algorithms are the following [38]:

- Geographical Routing Protocols: Essentially for cars with GPS. The solution consists in using information about node positioning, or about neighbours in a specific zone.

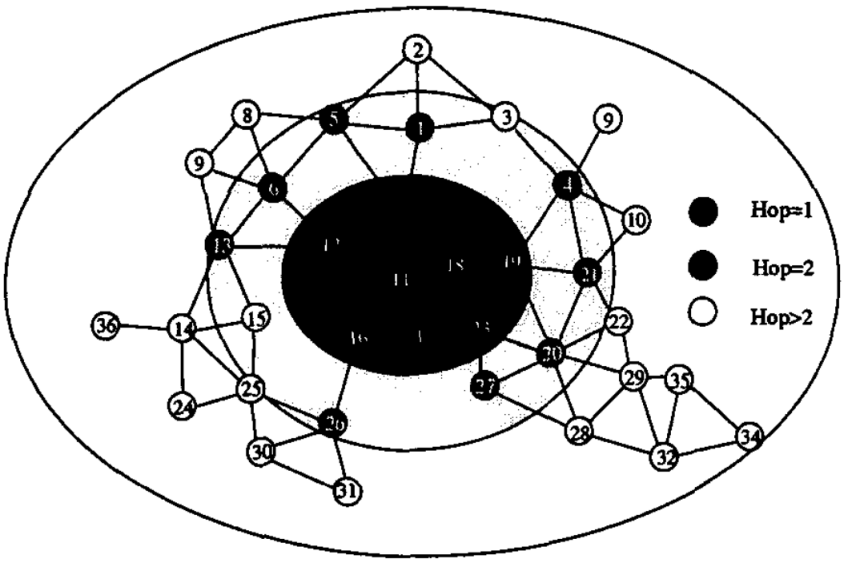



Figure 2.7: FSR scope [42]

- Movement-Based Routing Protocols: This solutions maintains memory of the trajectory made by the vehicle in order to calculate the velocities of the nodes, and with that information, choose the next best connection to make.

- Broadcasting Approaches: Used to disseminate information as fast as possible. This type of algorithms broadcast a message based on one or more factors (usually involving probabilities).

### 2.4.2 Sensor Networks Routing

In wireless sensor networks, a wide variety of protocols to gather data may be used. We will focus on three. The first type, may be branched in two options which are address centric or data centric. The second type, is based on a hierarchy (defining clusters, which have a relation like master-slave, as for example in LEACH [13]). The third type, is based on location (for example the Geographic Adaptive Fidelity (GAF) [46]).

Address centric, tries to find the shortest way to the sink, and the data centric tries to agglomerate all data in a common node, in order to minimize the number of packets sent, as seen in Figure 2.8.

The second type, works by defining masters in the network, which will be responsible

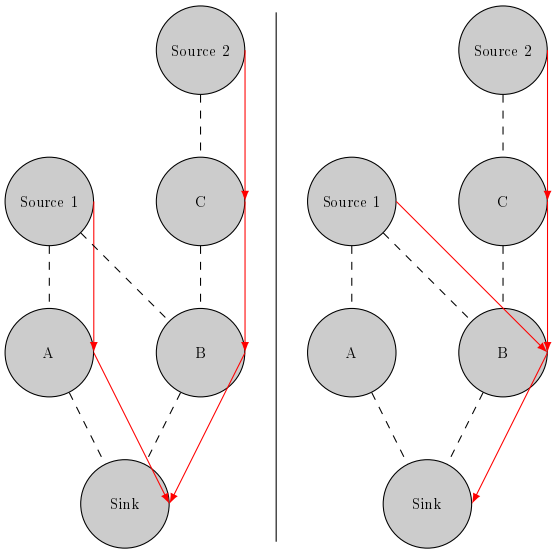



Figure 2.8: Address centric on the left versus Data centric on the right. Dashed means connection between nodes and arrowed means traffic flow.

for gathering the data from the slaves. Then, the masters are responsible to send the stored information to nodes closer to the sink.

The third type, is similar to the second type, but grids are established in the topology and masters are chosen depending on the location where they are in that moment.

## 2.5 Security Challenges in VANETs

Systems designed to operate in VANET should consider the security challenges present in these networks.

Some of the most crucial challenges to non Intelligent Transport System services are listed next [38].

- Data consistency liability: This is a important issue, since even authenticated vehicles can become malicious by sending incorrect information to gain advantage or disrupt the network.

- Key Distribution: For most security architectures, keys are essential to use some security protocols. However, keys cannot be installed by manufacturers because there is no cooperation between them regarding this matter. This would require agreement on the method of distribution.

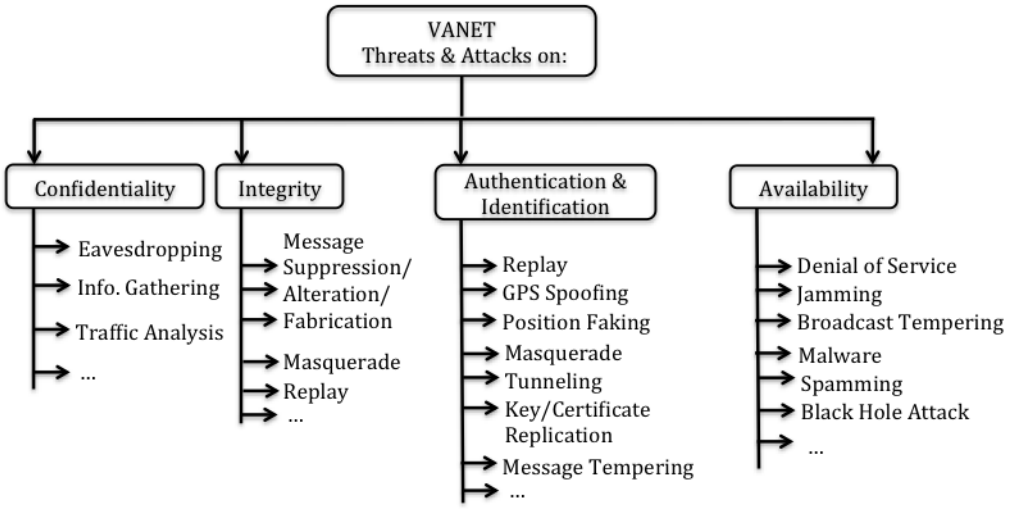



Figure 2.9: Attacks on VANETs [23, 35]

- High Mobility: The vehicle's computing platform, differ significantly in mobility support and throughput when compared with other computational systems. This creates a gap between fixed infrastructure and vehicular infrastructure, that may be closed by choosing lightweight security measures, without loosing the robustness.

## 2.6 Security Requirements in VANETs

Despite the various benefits offered by VANETs, securing this environment is very difficult, because of the issues that this type of network brings up (some listed in Figure 2.9).

According to ETSI' technical report on security requirements [11], the following objectives are defined:

- Confidentiality: In the absence of a confidentiality mechanism, this attack is very difficult to detect because it is passive, and users are not aware of data being collected. Because of that, and because this dissertation aims to deliver content between the user, and the external network securely, a confidentiality mechanism must exist.

- Mutual authentication, authorization and access control: Especially a requirement for non Intelligent Transport System (ITS) traffic, this requirement specifies that a node in the VANET realm must authenticate in the service provider, and also the server must be authenticated by the node, in order to prevent Man-in-the-Middle (MitM) attacks.

- Availability: Availability guarantees that the network is functional. An example of these attacks, is the Denial of Service (DoS). This type of property can never be totally guaranteed. However, the ways in which a third-party can disrupt the network availability may be limited.

- Authenticity: Authenticity is a major challenge for VANETs. Any node who acts on the network without authentication, can expose the network to serious consequences. This requirement purpose is to identify a node on the network, and deny service to non-authorized nodes.

- Accountability: This is the same as non-repudiation. The objective of this property is to identify vehicles injecting malicious data on the network.

- Integrity: Data integrity means that the accuracy and consistency of data must not be allowed to be tampered with. This requirement is applied both to ITS, and non ITS traffic. In practice, integrity control and authenticity are assured in the same way, because the origin will not be able to be identified if the content is altered.

It is important to realize that some attacks in VANETs, such as attacks on the technology availability cannot be prevented. An example of one of these attacks is jamming. If the technology becomes unusable, the only way to get communication, will be to change access technology, for example, by using cellular. In the eventuality of such an attack, the solution will be the versatility between technologies and not the protection on a specific one.

## 2.7 General Security Overview

This section presents several security primitives used in the dissertation that follows.

### 2.7.1 Security Versus Performance Versus Complexity

A system should not be deemed secure only because it is very complex, or because it relies on obscurity. A study on the dangers of complex systems to obscure security procedures [21, 48] revealed that complexity correlates to more frequent incidents. Not only that, but as security is usually based on mathematical or logical operations with keys, every measure taken in order to get security will add overhead to the system.

Because of the reasons mentioned before, systems should be designed to take security in account without adding an enormous toll on the functionality and performance of the system. When not doing so, systems will most likely have security incidents which may be hard to detect or fix.

### 2.7.2 Random number generation

Random number generators are widely used in cryptography in order to provide material to cryptographic primitives. They are especially useful for key generation.

An example of the problems that bad random number generation was shown in 1994, when a researcher from CERN Web team reported to Netscape that their version of Secure Sockets Layer (SSL) encryption protocol was seeded with the time of day, the PID and the PPID of the application using it.

Shannon entropy is a measure of unpredictability of the symbol a source outputs. Secure systems must have highly unpredictable streams, because if attackers can guess the sequence used to do operations on a buffer, they can mimic outputs.

In Linux there are two built in devices which are used as sources of random data:

- `/dev/random`: This device is a blocking random number generator. It blocks because the kernel and user-space need to collect information about events in the system to generate randomness, and the device blocks until a certain level of entropy is attained. It is especially useful for cryptographic purposes, however, it might not be possible to use this device if the system has to read a lot of random data. This is considered a "true" random number generator.

- `/dev/urandom`: This device is a non-blocking random generator. It is a pseudo-random number generator, because it takes the pool of randomness managed by the kernel, and swaps it, so that the order of the stream read is changed. Despite not being considered a "true" random device, "looking random" is enough for the large majority of the existent cryptographic protocols. In that aspect, this device is perfectly fine for giving unpredictable data, whilst `/dev/random` should be used for more demanding purposes.

A simplified scheme of how the random devices work in UNIX is shown in Figure 2.10.

The manual for the randomness devices has very vague description regarding the used sources of entropy [55], some of the listed ones are, for example:

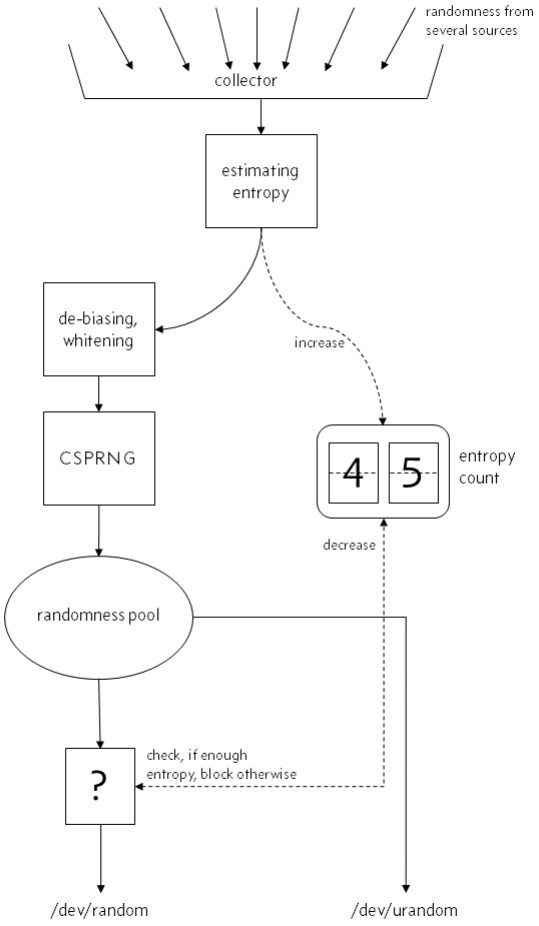



Figure 2.10: UNIX Random devices [20]

- Timing delays
- Devices inputs
- Interrupts

Every time the device is dumped, the system keeps track of how many bytes were added/removed, to maintain a certain level of entropy.

There are tests to measure the quality of randomness devices. The most used, is the Chi-square Test [3], which is largely used when sampling is large and discontinuous. This test is extremely sensitive to errors in pseudorandom sequence generators. The percentage is interpreted as follows: if the percentage is between 10% and 90%, the device may be considered random. The closest from 50% the better the device is.

After running the Chi-square Test on the `/dev/urandom` with a sample of 100Mb using a laptop with a i7-2670QM CPU @ 2.20GHz and using a Ubuntu v14.04.2 LTS with kernel version 3.13.0, the results were quite satisfactory (especially when knowing it's a pseudo-random device which does not maintain the level of entropy) by giving a percentage of 50%. Because of this, we can conclude that this device is better suited to time-constrained cryptographic operations, not only because it is not going to block the application, but also, because it is already considered quite random.

### 2.7.3 Cryptography

Cryptography is not a solution for the majority of the security problems. It might be part of the solution, but by itself, it does not provide any security. Let's say a person wants to encrypt a file with a key. Usually good keys are too long for memorizing and it needs to be stored somewhere. If the key is stored on same place where the file is located, then it defeats the whole purpose of applying the encryption, because if a attacker is able to get the file, it will probably also be able to get the key. Therefore, cryptography is part of the solution but by itself requires the use of other methods to protect the data (and the keys).

Assuming that keys can be safely stored, then it is a good measure to apply methods to hide and/or control the integrity of data.

#### 2.7.3.1 Encryption

As shown in Figure 2.11, if Alice wants to communicate with Bob over an insecure communication channel (in general, most communication channels are considered insecure). If no method for privacy is used, Eve (eavesdropper) can easily listen to the message. This may not be a big issue if, let us assume, the data is just a harmless conversation. But if Alice and Bob represent corporations, and they are exchanging confidential data, it turns out to be a big problem.

Therefore, to solve this problem, Alice and Bob may use encryption (shown in Figure 2.12), in order to transform the data they want to exchange, and hide the information from Eve, by using a symmetric key, for example. The question which arises is: how to exchange keys between both parties?

The idea is that Alice, uses a encryption function with a Key and a Plaintext as input, and the function returns a Ciphertext. Any changes in the Plaintext will change the result and the function is only reversible by using that key. When Bob receives the Ciphertext,

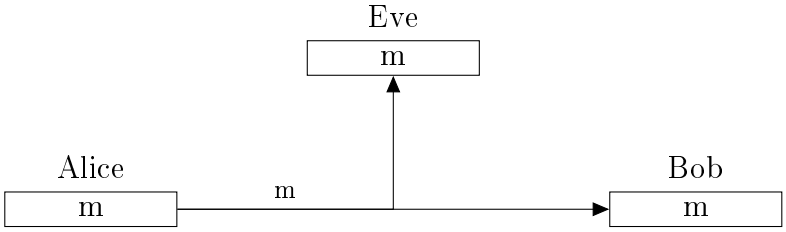



Figure 2.11: Insecure communication over insecure medium

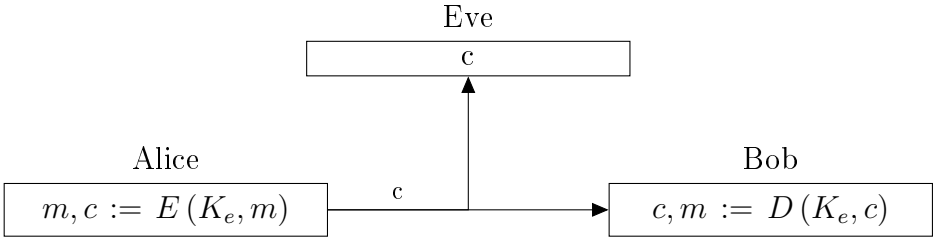



Figure 2.12: Confidential communication over insecure middle

---

he uses the same function but on the opposite direction to get the original Plaintext.

An attacker should not be able to learn any information about the contents of the message except the time sent and the recipients. Encryption, however, by itself does not hide the length of the packet, however for that purpose, a set of standards called Public-Key Cryptography Standards (PKCS) were created, and based on the PKCS#7 [24], it was created the Cryptographic Message Syntax (CMS) [18] which specifies the syntax for digitally signnatures, digests, authentication messages and encryption of data.

The CMS standard for encrypted data padding can be seen in Algorithm 1. The main idea behind it is that one should always append padding to adjust a buffer to the block size used by the encryption mechanism, and if the length of the packet is already adjusted to it, then it should be appended with *blocksize* bytes.

On decryption, the padding is also verified in order to confirm integrity. If the packet does not follow the established rules for padding, it shall be discarded.

Since the cryptographic routine is assumed to be unbreakable (if that is not assumed, then padding is still reversible), padding does not help security. However, this procedure makes traffic analysis much harder since it hides users' traffic true length, and makes attacks based on ciphertext and plaintext models much harder.

**Algorithm 1** CMS proposed algorithm

**Description:** Pad a buffer with size *length* according to CMS standard
**Require:** $length \geq 0$

```
1: procedure PadData(buffer, length, blksize)
2:     padding ← blksize − (length mod blksize)
3:     while i ≠ padding do
4:         buffer[i] ← padding
5:         i ← i + 1
6:     end while
7:     length ← length + padding
8:     return buffer, length
9: end procedure
```

#### 2.7.3.2 Authentication

Considering again the scenario depicted on Figure 2.12, Alice and Bob still have a problem, because even though Eve cannot see the packet contents, she may still interfere with the communication for example by:

- Insert new messages;
- Tamper with existing messages;
- Replay messages

Even if a message could only be understood by the key-holders, nothing guarantees Bob that the message came from Alice, Eve can send packets to Bob identifying herself as Alice (as seen in Figure 2.13 and Figure 2.14). To solve this problem there are several approaches, for example, digital signatures, which are based on Public-Key Infrastructure (PKI), Authenticated Encryption with Associated Data (AEAD) or Message Authentication Codes (MACs).

### 2.7.4 Key Exchange Protocols

#### 2.7.4.1 Diffie-Hellman (DH)

Diffie-Hellman (DH) was the one of the first asymmetric solutions to key distribution problem, it allows two parties, which have never met, to exchange a symmetric key over an insecure channel, without the possibility of getting the private session key derived by listening third parties.

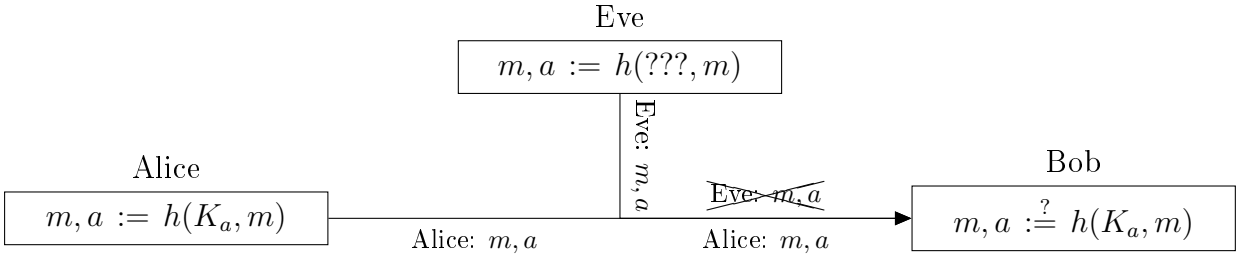



Figure 2.13: Authentic communication over insecure medium with MAC, computed as a Digest after the message m and the secret key $K_a$

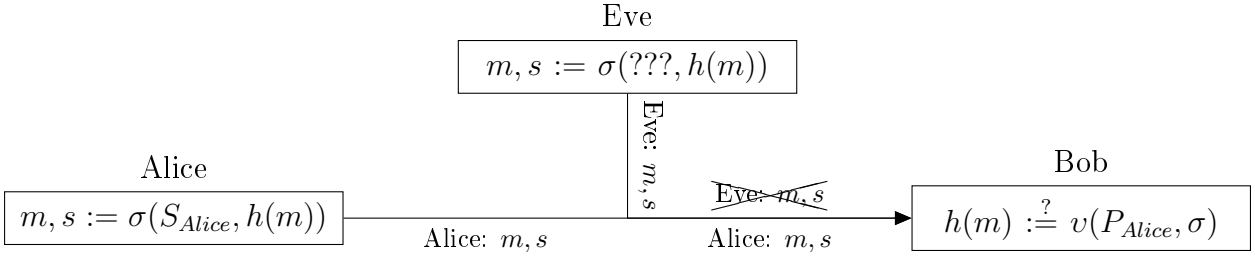



Figure 2.14: Authentic communication over insecure medium with PKI, using digital signatures generated with $S_{Alice}$ and verified with $P_{Alice}$

---

This protocol is, by itself, unauthenticated, which means it might be vulnerable to MitM attacks. To solve that the packets exchanged through the protocol must be signed.

The DH protocol goes as refereed in Figure 2.15.

The proof of DH can be demonstrated as follow:

$$K = (Y_x^y) \bmod q = (\alpha^x)^y \bmod q = (\alpha)^{x \times y} \bmod q \tag{2.1}$$

$$K = \left(Y_y^x\right) \bmod q = (\alpha^y)^x \bmod q = (\alpha)^{y \times x} \bmod q \tag{2.2}$$

Since the Equations 2.1 and 2.2 result in the same key, and cannot be calculated without knowledge of at least one of the private parameters, we reach to the conclusion that the algorithm succeeds in distributing a private key between two parties.

At the end of the protocol, an eavesdropper cannot get $K$ because $K = (\alpha)^{xy} \bmod q$, and there is no easy way to get this value without solving Discrete Logarithm Problem (DLP),

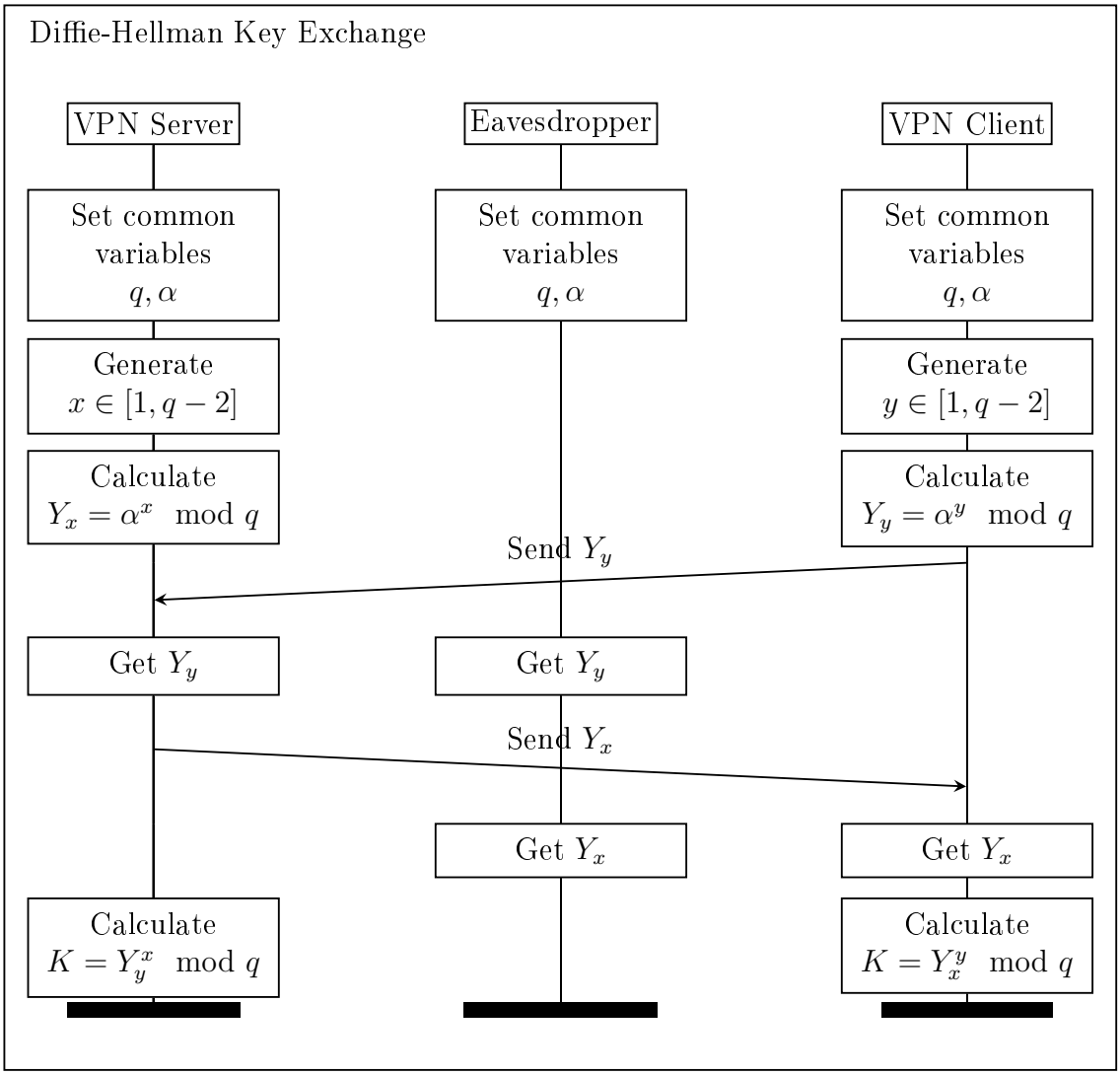



Figure 2.15: DH Key Exchange Protocol applied to a VPN key exchange protocol

or Diffie-Hellman Problem (DHP).

The key can be used in symmetric algorithms, which are usually more efficient than asymmetric algorithms.

### 2.7.5 RSA public-key schemes

RSA public-key schemes, are used to sign messages, and may also be used to encrypt them, but as the latter is a rather costly operation, when compared with most symmetric key algorithms, this work focus on the signing component because it is not frequently used.

The purpose of signing messages is not to make the message confidential, but rather, to assure that the content is trusted and immutable. A good analogy would be sealing a envelope with a specific wax seal. Everyone can read the message, but the fact that the packet contains a valid signed hash of the message means that the message was not changed since the signer signed it.

The process of signing a message involves generating a digest of the message, and use the private key of the signer to encrypt such digest. Since everyone can have access to the public key, the message is not secret, but after processing the signature with the public key of the signer, the hash calculated by the signer can be compared to the hash of the message which is being checked, if those hashes do not match, it means the message has been tampered with, or the origin is not trusted.

The RSA is a trapdoor function, which is a mathematical operation that is very easy to calculate in one direction but in the opposite direction is very difficult. For example $f(x) = x^e \mod n$ is easy to calculate but for its inverse to be calculated, the number has to be factored, which for large numbers, would be very hard using normal computational resources.

### 2.7.6 Hash Functions

The Secure Hash Algorithms (SHAs) is a particular implementation of a family of hash functions published by the National Institute of Standards and Technology (NIST). These algorithms are iterative, one-way functions which can reduce messages with arbitrary length into messages with a fixed length in such a way that having the hash cannot provide the original message but the message always gives the same hash.

These algorithms have several uses, but in the current context, they are used to verify message integrity: the smallest change in the message will alter the hash result completely.

A hash function must have 3 properties:

1. *preimage resistance*: For every output, it is computationally infeasible to find a input which yields the given output;

2. *2nd-preimage resistance*: It is computationally infeasible to find any second input which yields the given output;

3. *collision resistance*: For every output, it is computationally infeasible to find two distinct inputs which hash to the same output.

A hash function can be implemented with a scheme based on Figure 2.16. In the figure, it is shown that a digest is separated in fixed-length blocks and appended applying padding to match a size multiple of the block length. The compression function, and the optional pos-processing function depend on the algorithm used [36].

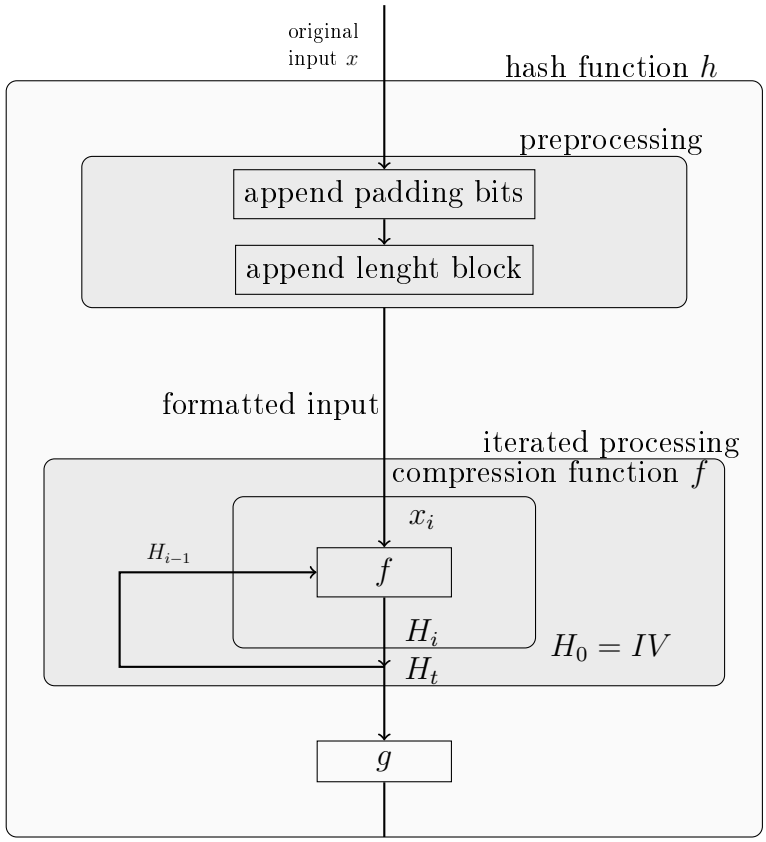



Figure 2.16: Merkle–Damgård Hash function construction

### 2.7.7 Symmetric Cipher Algorithms

Ciphers algorithms are a large set of mathematical functions, which transform a given plaintext into a ciphertext and also revert that operation.

The alternative from block ciphers are stream ciphers which typically require less complexity to implement and run, being for that reason, more time-efficient. This cipher type is however more tricky, because in order for it to be secure, the random number generator used to mask the bits must absolutely be unpredictable and unreadable from outside the operation and it's period should be large.

#### 2.7.7.1 Rijndael

Rijndael [43] is ranked first among the Advanced Encryption Standard (AES) submissions. This algorithm is not based in Feistel networks, instead it uses a substitution-permutation network, and is fast both on software and hardware.

The bytes are collected in a 4x4 matrix called state and the key size used specifies the number of rounds made by the algorithm.

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

The algorithm consists of the following procedures:

1. KeyExpansions: Operations on the key using the Rijndael key schedule.
2. InitialRound: AddRoundKey: Consists of a XOR of the subkey derived in the last phase with the state.
3. Rounds (repetitions according to key size)
   (a) SubBytes: Applies a S-Box to each byte in the state matrix.
   (b) ShiftRows: Cyclic shift of the last three by bytes of the state matrix.
   (c) Mix Columns: Applies a mathematical operation in each column.
   (d) AddRoundKey
4. FinalRound.

(a) SubBytes.

(b) ShiftRows.

(c) AddRoundKey.

There are no feasible attacks on full AES. There is one theoretically key-recovery attack which would take billions of years to complete [5]. There are also side-channel attacks which use measurements from the physical system to recover the key.

In terms of performance this algorithm was chosen by NIST by its high speed and low Random Access Memory (RAM) requirements. The algorithm requires 18 cycles per byte [49], and with AES-NI instruction set extensions, the throughput can be over 700MB/s per thread [34].

### 2.7.8 Modes of operation

The modes of operation of a cipher is a algorithm which establishes how a block should be processed depending on blocks before. It works with blocks or streams. In blocks a cipher algorithm is applied each block whilst on stream ciphers a random stream is XORed with each plaintext symbol.

The most simple one is Electronic Code Book (ECB) which ciphers blocks of fixed size independently of each other. This method is very weak because if data has a pattern, that becomes apparent for a person observing that cryptogram. By doing that, one can figure out the block size, and even the plaintext using frequency analysis.

The next method, Cipher Block Chaining (CBC), uses feedback to change the output of the next block with the previous.

Then there are several complex cipher modes such as the Output Feedback (OFB) and the Cipher Feedback (CFB) where a block cipher is turned into a continuous cipher by generating a random stream of blocks which would be XORed with the input. The difference between these two is in the feedback function, however, they work in a similar way.

The Counter (CTR) mode works is a intermediate step between ECB and CBC. After establishing a Initialization Vector (IV) which is public, each block is transformed by XORing the data with the IV plus the block number.

None of the modes of operation outputs anything which might be used for integrity control and because of that, other techniques such as Hash Message Authentication Code (HMAC) should be used.

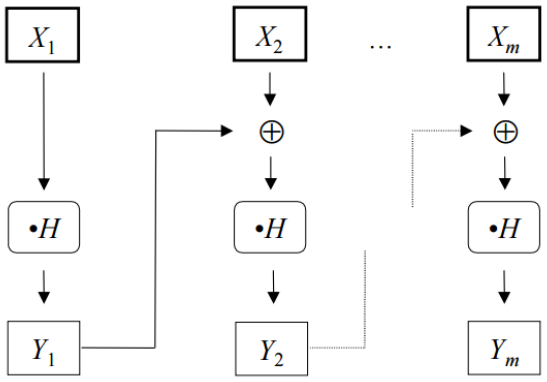



Figure 2.17: GCM Mode of Operation [39]

There is another mode of operation which is worth talking about in this document as it will be used in the implementation of the software.

Galois-Counter Mode (GCM) is a mode of operation which is used on block ciphers such as AES which provides authenticated encryption. This mode of operation is good for data exchanged in packets, because it can overlap the operations of authentication and encryption.

Basically, after ciphering each block it generates a Hash, which is used to cipher the next block as seen in Figure 2.17. In the last block, that Hash is returned and is considered a authentication tag because it is unique for each datagram.

### 2.7.9 Virtual Private Network (VPN)

The concept of VPN is widely used to denominate secure communication. Virtual Private Network Consortium (VPNC) defines that these solutions, should be categorized in two different types: trusted VPNs and secure VPN (and then it is possible to have a secure VPN over a trusted one to create a hybrid solution).

Trusted VPNs, assure that a given circuit was leased for a client and no one would use it.

Later, the Internet popularity started rising and, as vendors realised that trusted VPNs did not offer security, they created protocols that allowed to assure data confidentiality and authenticity, starting to develop solutions which are denominated by Secure VPNs. These types of solutions, do not have the necessity to lease circuits, instead the network is fully constructed on the endpoints, which are the stakeholders.

According to the VPNC, a VPN to be considered secure has to meet certain requi-

sites [8]:

- All traffic on the secure VPN must be encrypted and authenticated
  - VPN are used to establish a level of security, so that, a client must be authenticated and his traffic should be private. If it does not provide both these characteristics, it does not meet the requirements to be called a VPN.
- The security properties of the VPN must be agreed to by all parties in the VPN.
  - A VPN server might have to manage one or more tunnels. For each tunnel this property must be hold true for both endpoints.
- No one outside the VPN can affect the security properties of the VPN.
  - Third-parties should not be allowed to affect the secret keys of the stakeholders.

#### 2.7.9.1 Point-to-Point Tunnelling Protocol (PPTP)/Point-to-Point Protocol (PPP)

Point-to-Point Tunnelling Protocol (PPTP) [16] is used to create a VPN by relying in the Point-to-Point Protocol (PPP) protocol [50] to handle security between endpoints. This protocol instantiates a Generic Routing Encapsulation (GRE) [17] tunnel which will encapsulate PPP packets, as seen on Figure 2.18. This last encapsulation carries user data, and is not limited to IP, but it can carry also other protocols, such as Internetwork Packet Exchange (IPX) or Network Basic Input/Output System (NetBIOS). The control channel is instantiated in a side channel, running over Transmission Control Protocol (TCP).

PPP for authentication uses several mechanisms to authenticate interlocutors. Point-to-Point Authentication Protocol (PAP) [31], Challenge Handshake Authentication Protocol (CHAP) [51], MS-CHAPv1 [58], MS-CHAPv2 [57]. All these methods use passwords or challenges to authenticate users and they do not exchange keys.

The PAP, seen on Figure 2.19, is considered to be inadequate, since the username and the password are sent in clear text. The CHAP protocol already hides the username and the password, as seen on Figure 2.20. The MS-CHAPv1 is an attempt to improve security by storing a digest of the password and not the cleartext version of the password. The MS-CHAPv2 provides mutual authentication to the v1 protocol, by piggybacking a challenge on the Response packet, and an authenticator on the success packet.

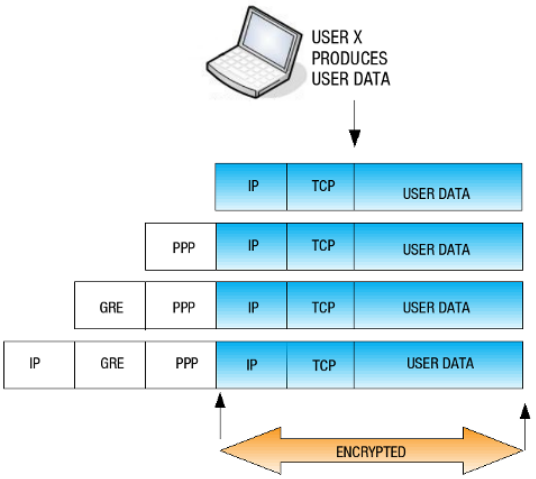



Figure 2.18: PPTP tunnel [2]

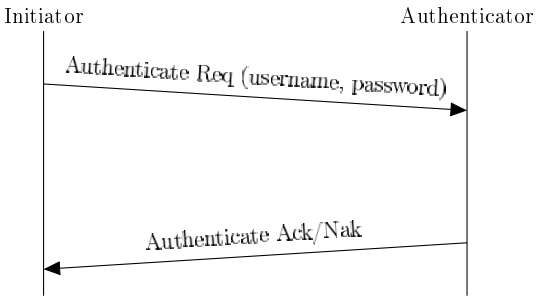



Figure 2.19: PAP protocol

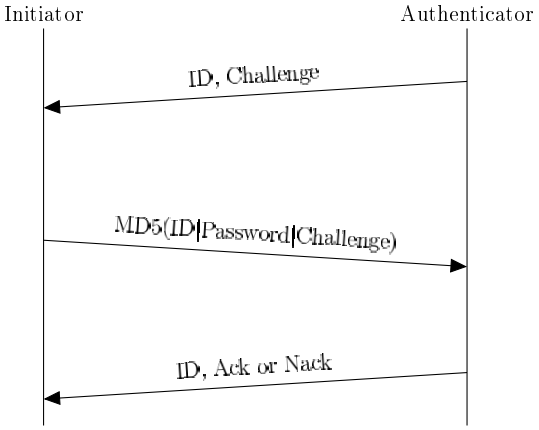



Figure 2.20: CHAP protocol

---

The encapsulated overhead of the traffic is the sum of the GRE header plus the PPP header, which is, at least, 14 bytes.

The lifetime of this solution, is the duration of the TCP connection on the control side-channel, which makes this unsuitable to use in this networks because in these environments, TCP connection breaks very easily.

#### 2.7.9.2 Internet Protocol Security (IPSec)

IPSec [29] is a security extension to the IP protocol. This extension uses Security Associations (SAs) structures to describe how communication should be secured. They hold an identifier, the Security Parameter Index (SPI), and the security mechanism used, the Authentication Header (AH) or the Encapsulating Security Payload (ESP). These SA are unidirectional and, in order to get bidirectional communication, two of these structures need to be instantiated, for each header used.

The protocols used to set these structures are the IP Security Protocol (IPSP), and the Internet Security Association and Key Management Protocol (ISAKMP). The last one provides a framework for authentication, which is key exchange independent and provides the Internet Key Exchange (IKE) [26] protocol and the Kerberized Internet Negotiation of Keys (KINK) [47] to exchange key material. The latter, is a protocol similar to IKE but uses the Kerberos protocol to allow third parties to authenticate stakeholders. The IKE does not implement a strict operation described by OAKLEY [40].

The IKE works in two phases:

- The IKE phase 1, establishes a secure authenticated channel using DH key exchange. This authentication is assured by using pre-shared keys, signatures or asymmetric keys.
- The IKE phase 2, use the previously established secure connection, to negotiate the IPSec's SA. Usually, two SA are needed, for inbound and outbound traffic.

The IKE phase 2 always operates in quick mode, whilst phase 1 may operate in main mode or aggressive mode. Aggressive mode does not protect the identity of the peers creating the secure channel and makes the negotiation to be made in 3 packets, whilst the main mode does protect the identity but takes more more time.

In total, the IKE exchanges exchange 6 messages, as seen in Figure 2.21, if both peers support cookies, if one of the peers do not support this, it is extended to 8 messages.

There is also support for IKEv2 Mobility and Multihoming Protocol (MOBIKE) [10], which exchanges the same messages as IKE. However, this protocol allow to change the IP in the SA structure. Even so, there is loss of connection when handover is made, because

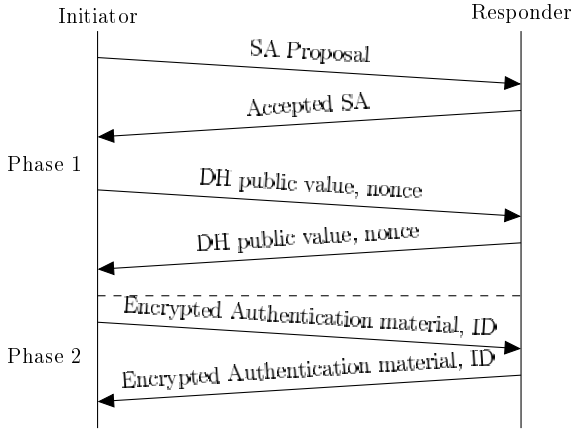



Figure 2.21: IKE main method

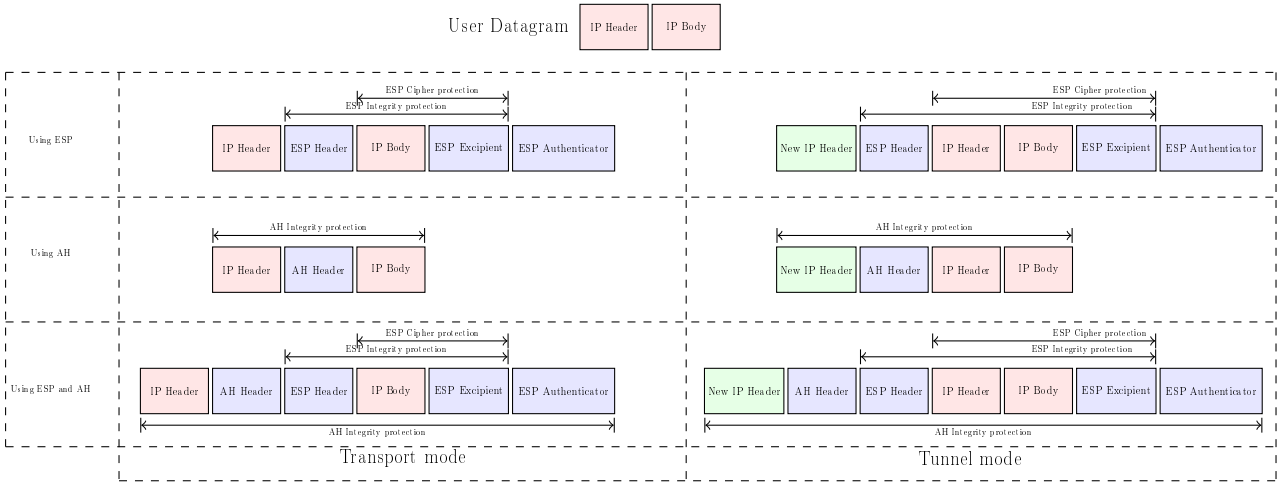



Figure 2.22: IPSec representation of the ESP and AH in the various IPSec modes [33, 59]

IP needs to be updated using a `UPDATE_SA_ADDRESSES` message, which needs to be verified for its authenticity.

IPSec works by using between one or two headers, the ESP [28] and the AH [27]. The AH does not encrypt the payload but it provides integrity check on complete packet. The ESP, however, protects only the payload encapsulated, both with encryption and integrity control. The AH, in order to support Network Address Translation (NAT) cannot do the integrity check on the whole packet, because source IP, for example, will be changed. In order to support this, if the value can be predicted, that value is inserted in the field for the Integrity Check Value (ICV) calculation, if it cannot be predicted, it will be set to zero. Rather than omitting the value, alignment is preserved. This process is repeated both in the sender, and the receiver.

IPSec can operate in two modes, as seen in Figure 2.22. The transport mode, uses the original IP address to route the packet and adds the security headers to protect payload. The tunnel mode, adds the security headers and a new IP address.

Transport mode is not usually seen as a VPN because are used in machine to machine communication, whilst the tunnel mode is used for machine to network or network to network applications.

This protocol could be used on top of Mobile IPv6, however, it would have a great amount of headers which are not needed, and since we can control the environment to create a meshed L2 network, this overhead could be removed. This makes the protocol not suitable, because it only supports IP and it sees both endpoints as equals, whilst we may want the server to impose conditions on nodes on the network.

#### 2.7.9.3 OpenVPN

OpenVPN [53] was developed as a versatile and portable solution, running in userspace.

The authentication uses SSL and asymmetric keys to authenticate both endpoints and uses the ESP to communicate securely between the stakeholders.

This solution also operates over IP, and supports both User Datagram Protocol (UDP) and TCP.

It can be configured in two modes:

- Routing mode: In this mode, the traffic of a user is routed through the tunnel in a IP-only network. This mode is more scalable than bridging because L2 broadcasts are replied by the OpenVPN administrative virtual interface, the TUN. However, it does not support L2 discover protocols and other L2 services.
- Bridging mode: This mode operates at a lower layer and creates a bridge between the user and the virtual interface, TAP. The performance is worse when compared with routing mode but it allows L2 services to be provided.

This solution is still not appropriate for this networks, since it runs over IP and does not support mobility, because security parameters are bound to IP.

## 2.8 Security Solutions for VANETs

The security solutions found in literature heavily depend on the uses given to the technology, for example, traffic safety purposes will not be spreading any potential confidential

information and because of that, encryption is not needed, but source authentication is very important in order to not fool legitimate vehicles with information from malicious nodes [45]. In opposition to this use of the VANETs, they can also be used to supply connection to services hosted on a provider network or even outside. These are two examples of uses which would use two completely different approaches in order to secure traffic.

The first use of this type of network could use, for example, Public Key approaches as described in [14, 19, 9], because it is a type of traffic which does not need to have high throughput, but rather, it should be authentic (at least the sender must be authenticated), as described in [45]. For this purpose, message signing is enough, and the overhead of using asymmetric techniques, which would in most cases make the network slower, is probably better than a handshake to establish a shared symmetric key which would be used only to exchange little traffic.

In the second case, the traffic would be approximately continuous and because of that, makes the previous solution infeasible because the number of times exponentiation would be applied to packets would, at a certain point, have harsher impact on the network performance and for this purpose, and this alone, a solution could be to establish a handshake with a server in infrastructure in order to establish a long term key to be used for all the traffic in a certain vehicle.

However, none of the documents really proposes a protocol for a secure infrastructure but rather, they discuss the use of some methods to authenticate a certain type of traffic.

Also, in [44] it is pointed out that symmetric solutions have a complexity of $\mathcal{O}(n^2)$, and asymmetric solutions have $\mathcal{O}(n)$. This is true when considering V-V communication. However, if we consider V-I, the complexity of the symmetric algorithms may become $\mathcal{O}(n)$, if the key is only established between the endpoints, hence solving the excessive memory consumption problem.

The big problem in asymmetric solutions, will always be the distribution of the key pair. How should the keys be put in a not (yet) authorized node and assure that they were not mislead, and the user ultimately using that private key is really authorized to do so.

The most reasonable way of doing this, since VANETs providing services in the cloud to users is always operated by a provider, keys may be pre-injected.

## 2.9 Chapter Considerations

In this chapter, we have explored several issues in VANETs, the cryptographic primitives, protocols and existent solutions which are able to solve those issues.

The design of security architectures in VANETs is of major importance, and the research in this field is almost non-existent. However, there are some published architectures that focus on traffic security applications, but largely rely on PKIs to assure node authentication.

Regarding the infotainment applications, the current solutions impose a large overhead, since they always rely on IP in order to establish a secure connection. Also, this type of networks are highly dynamic and the connection in them is unstable, whereby current security solutions do not apply.

In the following chapter, we will present a lightweight security solution, which was designed to be more reliable in cases of intermittent connectivity between nodes. Furthermore, the proposed solution aims to solve the protocol stacking overhead and apply authentication protocols in a intelligent way, without disregarding security. This will still provide a secure communication between the users, and the HN without introducing non-relevant layers.

# Chapter 3

# Link Layer Virtual Private Network

## 3.1 Introduction

In chapter 2 we have presented a brief overview of vehicular networks and security concepts. In practice, the current solutions deployed in static networks are not appropriate in VANET scenario. Therefore, in this chapter we propose a security architecture which can use some of these concepts to secure the user communication whilst supporting high mobility scenarios.

Section 3.2 describes the problem faced when considering the specific scenario of the vehicular networks. Here the current security solutions' problems are shown.

In section 3.3 is described a possible solution to the problem stated in section 3.2, as well as the stakeholders present in the following architecture and the communication requirements.

Section 3.4 overviews our solution, the stakeholders and the overall relation between them in the shown architecture.

Section 3.5 describes how communicating endpoints should be addressed in order to have a solution which is independent of the link layer technology used.

Section 3.6 describes the idea behind the VPN routing algorithm and a particular example of its implementation.

Section 3.7 describes how a vehicle is authenticated in a high level perspective.

Section 3.8 explains how users connected to authenticated access points are addressed.

Section 3.9 describes the structure of the VPN packets.

Section 3.10 shows the software architecture, the functional modules and the flow of different types of messages through each module and, the mechanism implemented to

prevent memory overflows.

In section 3.11 some problems found in the solution are shown and their solutions.

In section 3.12 the considerations about the architecture are described.

## 3.2 Problem Statement

The architecture in place uses devices to create a meshed network and communicate to external providers, these devices are described next.

- `RSU`: This device is an interface between the 802.11p Layer 2 meshed network and the Layer 3 network connecting it to the VPN server.

  The RSU is a fixed access point to the vehicular network, and it behaves as a gateway between the OBUs and the VPN server. This node provides interaction between the vehicular network, and external fixed networks.

- `OBU`: This device is a mobile node to which users can connect using well-known technologies, such as 802.11g. OBUs form a meshed network between themselves, forming branches (like a tree) with a starting point in the fixed infrastructure. They also have other ways to connect to the HN when the infrastructure is not available, for example using cellular. However, such approach is usually avoided since this technology is very expensive, when compared to Wireless Local Area Network (WLAN) technologies cost.

Since vehicles move, the security solution should tolerate fast and frequent handovers. This imply not renegotiating the secure tunnel parameters upon handovers.

Previously, some existing security solutions were described. Using Mobile IP (MIP), some of those solutions could be applied. However, they have some problems.

PPTP would require a constant connection because the negotiation of the tunnel and its maintenance is controlled with an initial TCP connection, which would break on connection loss. OpenVPN would work in UDP mode while the IP of the mobile nodes is maintained constant. However, this may not be assured in these dynamic environments.

IPSec supports mobility by using MOBIKE. However, it assumes that both peers can change their IP address by negotiating new asymmetric SAs. These operations can add some protocol complexity and aggravate the SA setup which is already cumbersome [38].

Furthermore, we want to implement VPNs to vehicles that may be identified by their own endpoint identifier.

## 3.3 Requirements of the Solution

We want user data flowing in the network to be protected. The vehicle should be authorized to access the HN through a meshed network, and the data sent and received by it should be confidential in between stakeholders.

To avoid the problems existing in previously presented solutions, the solution developed must be lightweight and exploit the VANET technology characteristics to create a dynamic, IP-independent, L3 mobile VPN. The solution should depend on a unique vehicle identifier which must be maintained constant at all times independently of the state of the connection or the access technology as in [60]. Also, because we have full control of the environment, when designing a solution like this, we can also omit the outer IP-layer, keeping only the protected inner layers belonging to the users and exchanging them between the vehicles and the server through a secure tunnel, using the VPN identifier to route data.

The security parameters should also be created only once, and maintained even when the node has no connectivity. This way, when the vehicle is in range, it may resume communication with the server, because they both have the security parameters already established, which prevents blackouts due to renegotiation.

The security parameters should be negotiated by the endpoints, and a secure tunnel should be established on top of a routing strategy. Furthermore, to provide a seamless experience, the security parameters should not depend on parameters changing when mobility occurs. For that purpose, and considering the available infrastructure, we propose another entity, the VPN server.

The VPN server establishes the required security parameters for each vehicle, identified as a unique Endpoint Identifier (EID), in order to provide access to the HN. This server may be located in the same network as the RSUs or be deployed in an external network.

A user, which might be considered a mobile node, uses an access point provided by an OBU to get a secure connection to a HN.

This secure connection encompasses a set of security requirements that should be met, as for example, authentication and confidentiality. In that sense, this secure connection resumes the requirements established for VPN.

The OBU may use several means available to connect to a HN, for example a direct connection to an RSU or a multi-hop connection to other vehicles connected somehow to the infrastructure.

The VPN server should authenticate each vehicle in order to grant access to the HN and manage traffic between this network and each OBU.

Each OBU should act as a Network Access Server (NAS) [37] to provide access between the user and a HN, through its VPN server. It will route user data securely to a HN and vice-versa.

Despite the authentication process of each OBU and the connectivity transparency for the user, handovers will occur when a vehicle gets a connection with a better metric to the infrastructure. Since the network will support multi-hop, the handover should be made between RSUs and between OBUs. The handovers should be transparent to the users when using an OBU as an access point, but not to users changing between OBUs.

Three security requirements established are the following:

- Users' traffic should be confidential between the OBU providing the access point, and their HN.
- Message integrity must be assured. Any modification to the traffic made by other persons than the stakeholders must be detected and rejected.
- Traffic between the HN and the OBU should not be diverted by attackers to other hosts in order to prevent against DoS and vice-versa.

Depending on the architecture configuration, the HN could either identify users connected to OBUs or just the OBU in which the user is connected to. The first situation is much more flexible because it allows to establish different HN access network policies for each user.

## 3.4 Network Architecture

The network may be divided in two different segments, as can be seen in Figure 3.1:

- The Ad-Hoc Meshed segment may be considered one big network. Since we are in full control of how traffic moves here, the VPN may be supported only by the link layer and a few headers which are mandatory such as an unique identifier.
- The Layer 3 segment, which is required because when leaving the network that we control, the traffic has to be in accordance with standards used by external network providers.

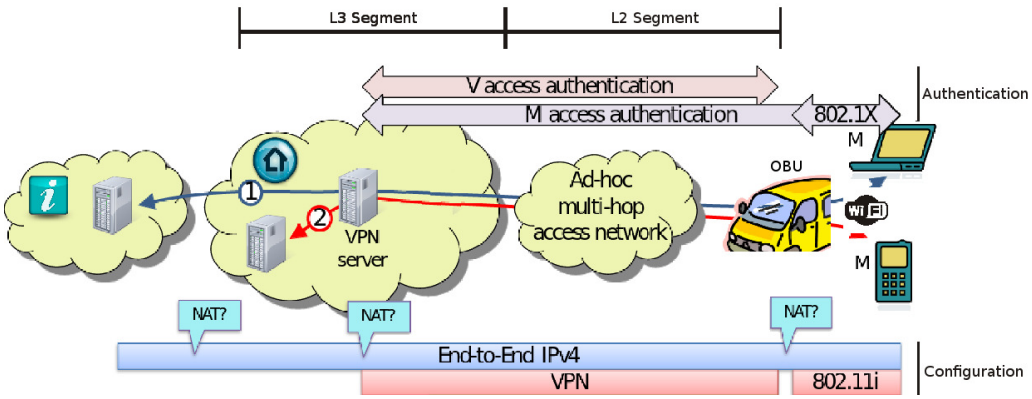



Figure 3.1: Proposed Network Architecture. Users (M) connect over a Link Layer tunnel formed between vehicles (V) and the VPN server

---

Users' equipment (M) connecting to the vehicles (V) are authenticated with standard technologies, using for example 802.1X. This is a well-known user authentication mechanism in Wireless LAN, and for traffic encryption the 802.11i (known as Wi-Fi Protected Access II (WPA2) which is widely used in most private wireless networks nowadays) should be used because it protects the entire L2 packet.

Vehicles will then establish a secure session with the VPN server, by using pre-deployed authentication credentials, allowing them to negotiate specific parameters that belong to that secure tunnel.

For already created sessions, in the VPN server, packet protection should be verified and reversed allowing users to access services within the HN. Since several devices may connect to a specific OBU, and they communicate using the TCP/IP stack, the server must be able to keep track of the IP of each device in each OBU, in order to forward packets accordingly.

The association established in the server should be kept at long term in order to reduce renegotiation to a minimum, and should not depend on the IP given to the user or the vehicle. This feature allows the VPN server to work even with different technologies and to re-establish the connection after handover with minimum latency.

This architecture allows several different NAT deployments. NAT deployed in the VPN Server will allow the traffic from the users to be undifferentiated on the HN services and in the HN gateway, NAT deployment at the HN gateway will allow to differentiate the users IP in services within the HN but not outside of the HN. These two layers of NAT are perfectly compatible. At last NAT could also be deployed at the OBU. This third

deployment could be more troublesome because it restricts communication between two different vehicles because it will make the server differentiate only vehicles and not users connected to them, which makes impossible for the VPN server to address user IP and therefore restricting communication between users in different vehicles.

## 3.5 Node Identification

The OBUs should not be differentiated by the address given by the access technology which they are using at the moment to address the external network. Instead, they should be identified by a Vehicle Endpoint Identifier (VEID) which should be unique in the network and independent of the interface in which it is receiving information. This avoids problems with IP mobility.

The usefulness of this VEID is not only restricted to the advantages it has in routing, but also to the authentication process. If authentication is bound to this unique identifier, the node will never need to renegotiate a VPN when the network interface card addresses change.

The users, however, use the normal communication stack, with IP. Because of that, mobility is provided to the vehicles but not to users changing access points.

## 3.6 Forwarding

Generally a routing mechanism should try to optimize one or combinations of the following situations:

- Shortest path (fewest hops)
- Shortest time (lowest latency)
- Shortest weighted path (based on several factors, for example bandwidth, signal strength, ...)

The solution proposes to authenticate nodes by their EID and also, to reduce the number of layers used by common protocols. Because of those are the goals, IP layer is then, unnecessary. A meshed forwarding protocol similar to service-based routing [32] was created, which identifies nodes based on their VEID, that also represents the tunnel. The protocol is then specified in this section.

1. RSUs periodically transmit beacons. Those beacons indicate direct access to the infrastructure. The lack of beacons in the environment will be interpreted by the nodes as being out of the infrastructure coverage.

2. OBUs keep listening to environment messages, establishing their uplink connection from the message with the best metric to the HN (be it the number of hops, RSSI, or any other combination). If there is a tie between the current uplink and another one received, the OBU will check for the RSSI and connects to the one which has a higher value. It is also possible to carry along the path the sum of the RSSI to the RSU and calculate the overall best connection by calculating the Equation 3.1:

$$\overline{RSSI} = \sum_{i=0}^{i=n} \frac{RSSI_i}{n}, \qquad n = \text{Number of Hops} \tag{3.1}$$

Every time an OBU refreshes its uplink connection, it sends a message (`SESSIONNOTIFY`) to the uplink advertising itself and broadcasts a `BEACON` message. By sending a beacon only as a reaction to another beacon, and not periodically, we can save bandwidth and processing, since we know that these message must start at the RSU, periodically, and if OBU cannot receive beacons, then it is not in the range of any RSU. When a OBU or a RSU receive a `SESSIONOTIFY` packet, the path to that node is added to the downlink table and that packet is then carried to the VPN server.

This mechanism aimed to establish the shortest path to the server. The algorithm tried to maintain the best connection to the server by establishing a path with the fewest number of hops. Since in some situations, the node would change it's routes very quickly (by having a tie in the number of hops) even if there was no advantage of doing so, a new factor was brought in consideration, the RSSI. With the addition of the RSSI factor, it was possible to solve ties in the number of hops and only change path if the new one had a better RSSI. This prevented the change to paths with the same distance to RSUs but worse channel condition (either because of distance or interference).

However, it is also added a hysteresis mechanism which is based on the RSSI to make decisions when the metric is tied, as shown in Figure 3.2. As shown in the figure, the route will not change immediately to a new connection if that new connection is not significantly better.

The route establishment decision process is described in Figure 3.3. Every node in the meshed network makes decisions based on this algorithm.

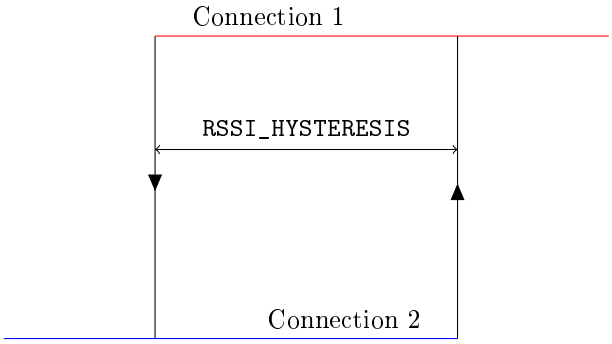



Figure 3.2: Hysteresis mechanism based on RSSI

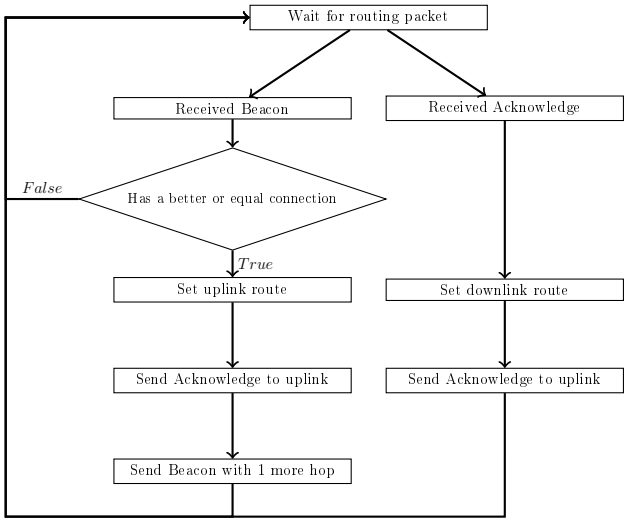



Figure 3.3: Node Forwarding Algorithm

The OBUs only need to know where the infrastructure is in order to communicate with the server. Also, each OBU knows which OBUs are transmitting data to the infrastructure through it.

The protocol is proactive, because it maintains a fresh list of paths to the destination and periodically updates the table. However, for it to work, the table does not need to contain every path but just the next-hop to the server and the next-hop to downlink. Instead of having the whole path as in source routing, or specifying the destination as in destination routing, the packet is marked with an EID and when it is sent to the network, the packet will follow a flow as seen in Figure 3.4. The figure shows that a packet sent by a OBU will always go upstream to the server unless the path is corrected in the middle (in this case the packet will reverse its flow until it gets a valid route, and helps nodes detecting invalid entries in the table).

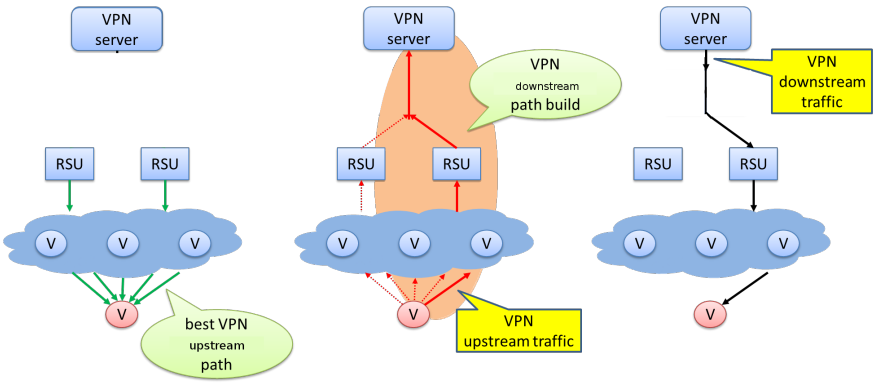



Figure 3.4: Routing Scheme

Since vehicles move, the downstream routes may become outdated. A fallback mechanism is implemented: when a packet reaches a dead-end, the flow that it is following is reversed until it reaches a valid route, removing all outdated routes, and then reversing the direction of the flow again in order to use correct routes as shown in Figure 3.5.

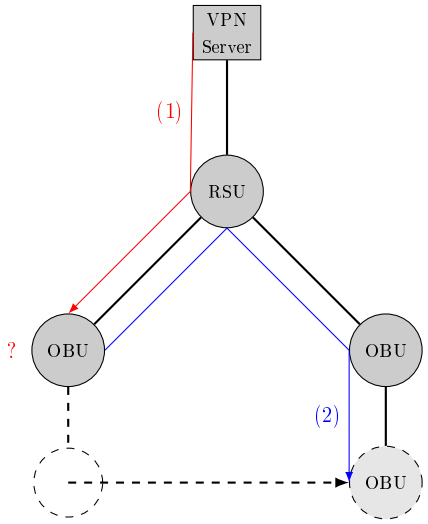



Figure 3.5: Fallback mechanism used to recover packets

In order to implement this routing scheme, structures are needed to hold the information relating the hop by hop information. This structure can contain the same information for both the uplink and downlink information. For each EID, the structure should contain the MAC address of the interface relating to the Network Interface Card (NIC) with contact to the next-hop and the MAC address of the next-hop to reach that EID. The structure should also contain the metrics used to decide, in order to be able to compare with the next values of the received possible paths and the timestamp when the last decision was

successfully made. On the server the only parameters necessary to be stored are the RSU IP and its port.

A diagram displaying what each node has to do in order to construct the forwarding tables is shown in the Figure 3.6 and is explained next.

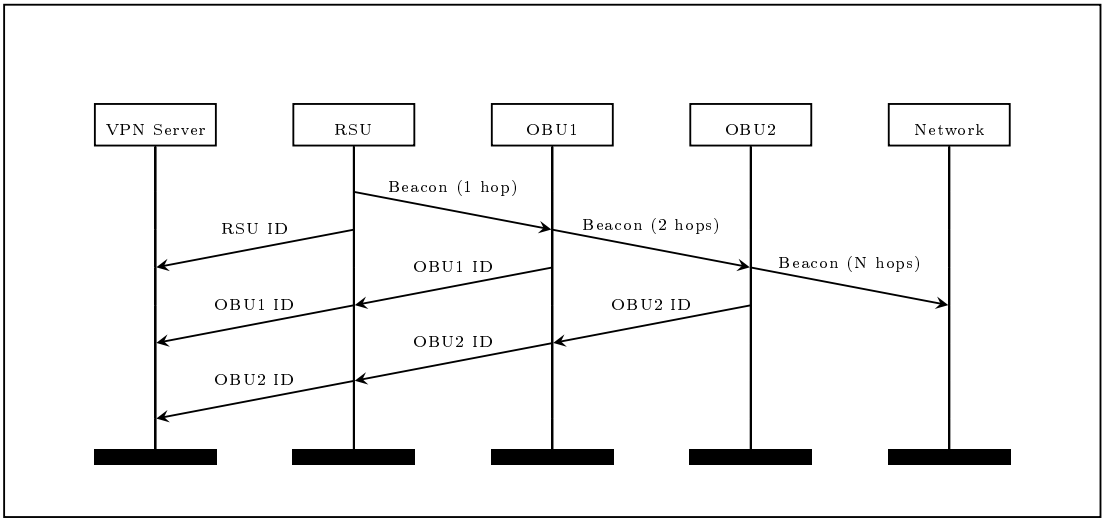



Figure 3.6: Forwarding Protocol

In this example, there is a RSU and two OBUs in the network. As described before the RSU will send periodic beacons with a metric, e.g. 1 hop. Because OBU1 is within the RSU's range, it connects to it, acknowledges the connection and broadcasts the service. OBU2 is only in OBU1's range, it connects to OBU1, acknowledges the connection and broadcasts the service. Because, in the example, there is no other node in range, the packet will not be retransmitted again, preventing network flooding. Since every node registers the connections being made, the tables in the nodes would look like shown in Table 3.1.

Table 3.1: Forwarding Tables

| **VPN server** | **RSU** | **OBU1** | **OBU2** |
|---|---|---|---|
| **Uplink** | **Uplink** | **Uplink** | **Uplink** |
| —— | —— | $RSU\ MAC$ | $OBU1\ MAC$ |
| **Downlink** | **Downlink** | **Downlink** | **Downlink** |
| $RSU \rightarrow RSU[IP, PORT]$ | $OBU1 \rightarrow OBU1\ MAC$ | $OBU2 \rightarrow OBU2\ MAC$ | —— |
| $OBU1 \rightarrow RSU[IP, PORT]$ | $OBU2 \rightarrow OBU1\ MAC$ | —— | —— |
| $OBU2 \rightarrow RSU[IP, PORT]$ | —— | —— | —— |

## 3.7 Authentication in HN

In order to be accepted in the server, all the intervening nodes must have its own private key and the server public key, with the exception of the server which needs only its private key and all the authorized nodes' public keys. If any vehicular node does not meet these requirements, it must not be able to establish a tunnel with the server. Any intervention regarding the key establishment by a third party in the network should also be detected and rejected. The keys should be pre-deployed and they should be protected against third-party modifications (copy and removal included).

In order to start the authentication, one of the endpoints has to initiate the key negotiation. If the server started negotiating the parameters, it would have to iterate over every node in order to check for key validity or even existence. Then, the OBU is the stakeholder which always starts the key exchange. However, if the OBU has a SA but the server is restarted, this approach will not be able to renegotiate new parameters. Because of that, the first time the server becomes aware of the vehicle, it asks that vehicle to start the authentication process (by sending `VPN_SETUP_START` to the node).

When the vehicular node receives a packet to start the authentication (`VPN_SETUP_START`), it starts a signed Diffie-Hellman (DH) key exchange which will generate a secret, shared key, that will be used by the ciphering algorithm to protect the payload of every user data packet exchanged between the server and the vehicle. The `VPN_SETUP_START` is not necessary to start the authentication. The node can also start the exchange, if the key expires.

The operation of this protocol can be seen on Figure 3.7, where it is shown the case in which the server became aware of an unauthenticated node in the network, therefore, requesting its authentication, afterwards the key lifetime passed and the node tried to generate new key material.

The amount of data signed in each packet is different. While in `VPN_SETUP_START` only the signature is hashed/signed because the crucial information behind it (the EID) is also included in the Signature header and it would be redundant to do it, in the other authentication packets the public key is added to the digest.

With those fresh-generated keys, the endpoints calculate a session key which may be used for a long time, and discard the other keys in order to achieve Perfect Forward Secrecy (PFS).

There is a time-out associated with the key, because now the only way to calculate the key is solving the DHP, or the DLP and that cannot be done currently faster than the

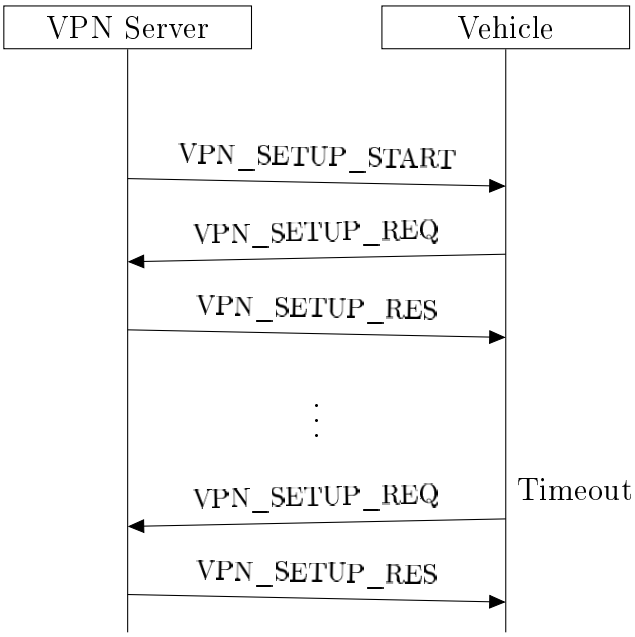



Figure 3.7: Authentication Protocol

timeout.

When exchanging data between the HN and the user, the symmetric key derived from the authentication process belonging to the correspondent vehicle is always used to protect the data and its integrity.

The key derived from this process is stored in a structure which has the same composition on both the vehicular node, and the server. This structure only stores the key, in a char array in order to prevent several dynamic memory allocations when using it, and it contains also the corresponding EID. Also, the timestamp is stored in order to maintain a record of the validity of the key.

The process in each node can be described by Figure 3.8. In the figure it is shown that each node has 3 states. The second one, `NEGOTIATING`, is a temporary state in which the node waits for the reply of the server. If the server does not reply, the node will fall to the `NOTSET` state or, if a reply is received and is valid, it proceeds to calculate session parameters and fall into the `VALID` state which is maintained until timeout or renegotiation request. Nodes always start on the `NOTSET` state and they are therefore, the ones to start the authentication protocol.

Also, as seen on Figure 3.9, when the server detects a new node connecting, it sends a signed packet requesting the node to start the DH key exchange. This only occurs once per node because if the node disappears, the VPN server shall not start the renegotiation. If a

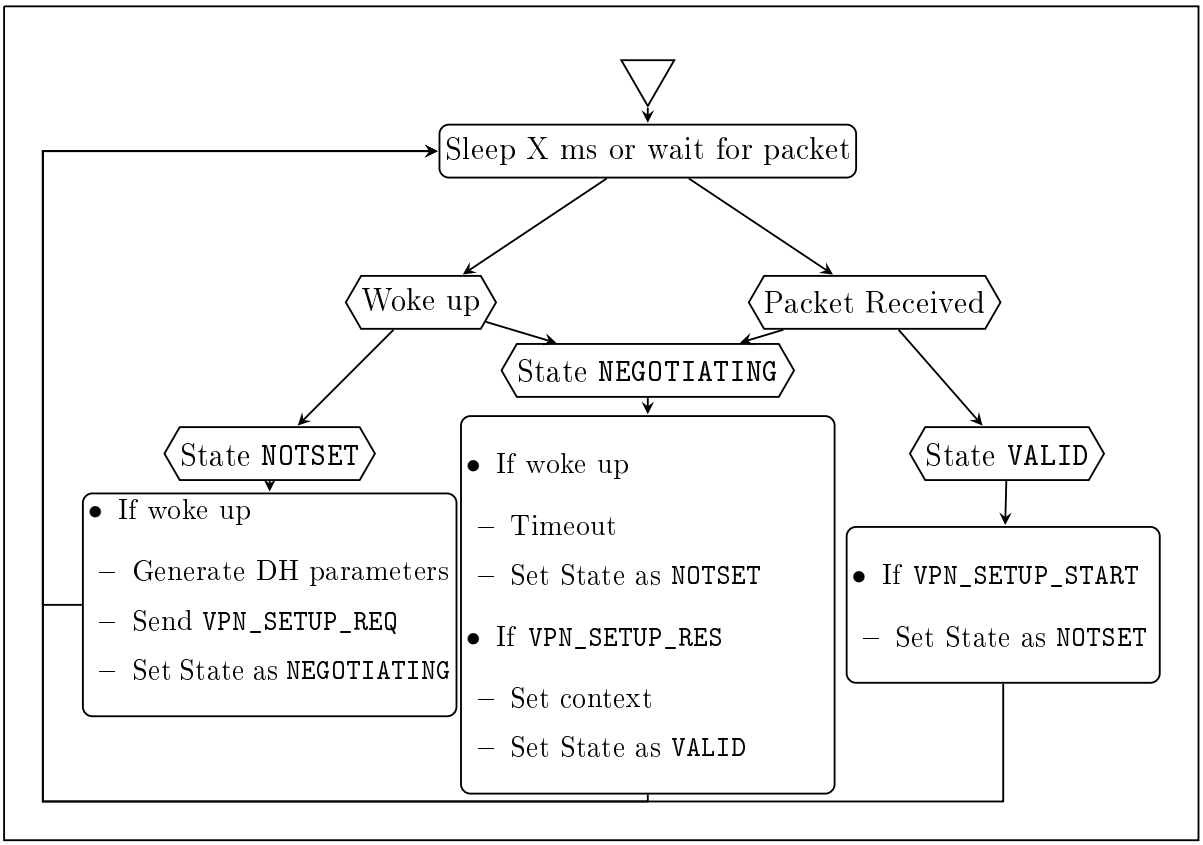



Figure 3.8: Node Authentication State Machine

timeout occurs, the node will start, by its own will, the DH key exchange with the server. All keys are registered in the server structure which maintain parameters for communicating with all nodes.

## 3.8 User Traffic Association

In the server, it is kept a list of the users' addresses in each node, so that the VPN traffic can be routed accordingly. In order to do this a structure was created which holds the EID and the lookup key is the EID. When the traffic comes from the user, the source IP is checked and that IP is associated to the EID where the packet originated. Furthermore, when the server wants to communicate with a user, it looks for the EID where that destination IP is connected and sends it only to that node.

By doing this, we can address not only nodes, by their EID, but also users connected to the access points by their IP.

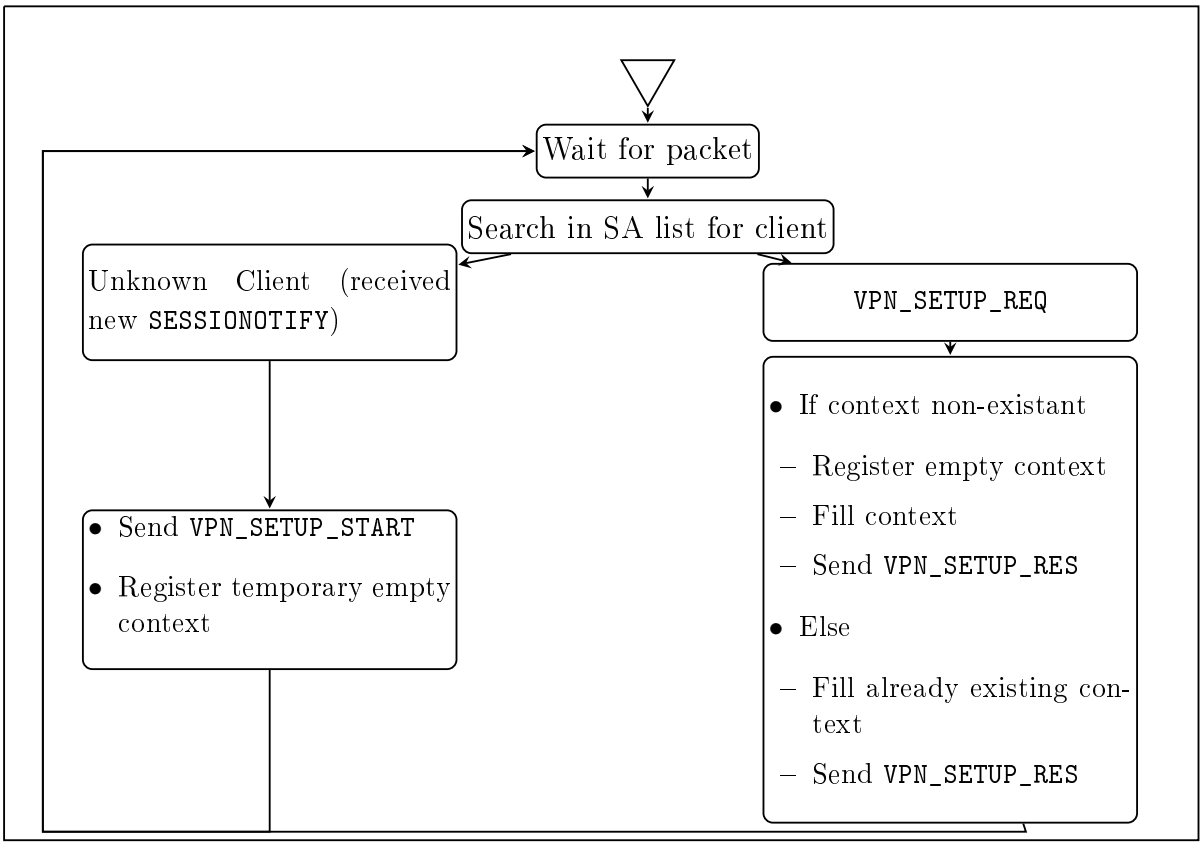



Figure 3.9: Server Security Association Management Algorithm

## 3.9 Packet Structure Overview

In order to reduce the overhead added by headers which are used in standard solutions, and since the meshed segment of the network allows to manipulate packets over L2, it was added a new layer on top of MAC, this header is used to forward packets between the VPN server and the vehicle.

The packet type, which may contain `BEACON` or `SESSIONOTIFY` for routing establishment packets, `VPN_SETUP_START`, `VPN_SETUP_REQ` or `VPN_SETUP_RES` for authentication packets or other numbers for other types of data. The `Metric` field will carry information about the main metric used in forwarding packets and in normal traffic will carry information about the direction of the flow which the packet is following (if the packet is following a good route, it is `NORMALD`, if not it is `REDIRECTD`). The EID field is the unique identifier used by the node.

The payload composition may vary depending on the `Type` of packet. During authentication, the payload contains the sender DH Public-Key and a Signature header shown in

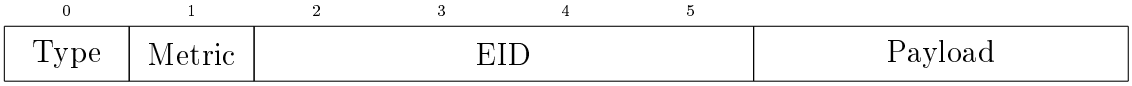



Figure 3.10: EID Header

---

Figure 3.11, which allows authentication using RSA public-key schemes.

The packets with `Type VPN_SETUP_START` only contains the Signature where the Digest is a one-way function of the other Signature fields.

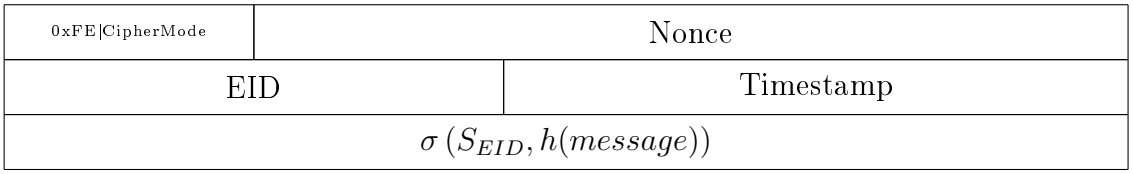



Figure 3.11: Signature Header: The *CipherMode* variable is a bit that indicates if the node is requesting cipher or not. The last field is a digital signature applied on the hash of the message

The packets with `Type VPN_SETUP_REQ` and `VPN_SETUP_RES` are composed by a runtime generated DH public key and the Signature header. The public key is signed in order not to be altered. By doing this, an active attacker will not be able to change the key whilst maintaining a valid Signature field.

As a note, standard network order for packet fields should be big Endian in every network exchange, and the sender must convert the fields which are composed by more than one byte to network order so that compatibility between different architectures is maintained. Similarly, the receiver must first convert the fields in the packet and only then use them.

User data is captured in the OBU, padded, ciphered with the current VEID key and then appended to a new packet containing a routing header.

## 3.10 Software Architecture

In order to implement the network architecture, the software is organized in modules. Each one of these modules has specific functions in order to detach each plane of the network, and also to separate the program operation in different stack levels.

The following modules were defined for low level interface with network devices:

- Transmission and Receiving modules for control plane.
- Transmission and Receiving modules for data plane.
- Transmission and Receiving modules for Layer 3.

The modules for Layer 3 interface are used in RSUs, where communication requires the support of external providers. The modules for sending and receiving data at Layer 2 are separated into two different planes in order to separate route establishment from data traffic.

These modules communicate with two other modules:

- The control plane manager module receives routing information from the network in order to construct its own routing tables.
- The data plane manager module receives data and authentication packets, and uses the tables established in the control plane manager to forward packets between nodes.

If the data packet belongs to a specific node, the authentication or tunnel manager modules will be called. These are the highest level modules. The authentication module manages the session state, generates and exchanges keys. The tunnel manager applies transformations to incoming and outgoing packets. However, some of the refereed modules do not need to be used in all the nodes in the same manner. For example, the authentication manager module, in the server , will never process `VPN_SETUP_RES` packets and the vehicular node will never process `VPN_SETUP_REQ` packets. Because of that, it does not make sense to accept those packets in those cases.

The VPN server acts as a gateway from the users to the HN and vice-versa. For that purpose it contains three fundamental components working with specific purposes according to Figure 3.12. The figure shows that in the server the only modules are the authentication manager module, the tunnel manager module, and the receiving and transmitting modules which interact with the other two.

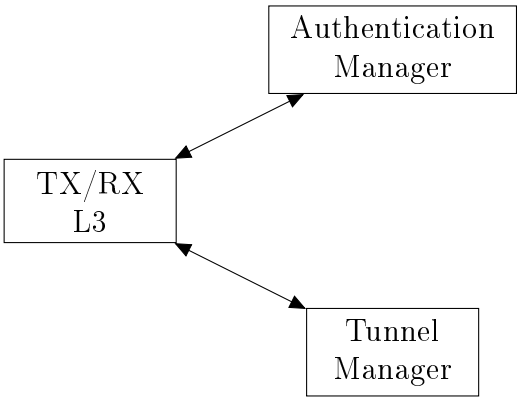



Figure 3.12: VPN Server Components

In Figure 3.13 it is illustrated an example of the working flow of the node. As seen on the figure, the receiving module manages the routes of each EID and may create contexts if the node is not known. When authentication data is received, that data will be passed to the authentication manager in order to give access to the node. When data packets are received, the tunnel manager will wake up and will process the packets.

As seen in Figure 3.14, the RSU is formed by the modules that, in cooperation, allow packets which are being transported over Link Layer to change and be transported over Network Layer. This can be done only by removing the MAC layer which is passed in the vehicular environment (in the tunnel manager), and append the rest of the packet after IP (in the L3 transmitting module). This is a requirement for this node because the server can be running on networks crossing other external providers. The authentication manager in the RSU is not necessary, however, it allows the RSU to authenticate in the server. The RSU manager module is the module that triggers the beginning of the routing process, by advertising the RSU to the server, and to the control plane forwarding manager. The control plane forwarding manager constructs the routing tables when it receives packets, and the data plane forwarding manager uses those tables to route data.

A high level description of how the modules communicate between them is shown in Figure 3.15. As seen in the figure, in the RSU can receive authentication information directed to it from the server and respond to it. For data that it is not addressed to it, it will append or remove the MAC layer in accordance to the situation, and send the packet to the meshed network if it is received from the server or send the packet to the server, if it is received from the meshed network.

The main difference between the RSU and the OBU is that the RSU sends packets going to the server always to the L3 low level interfaces, whilst the OBU keeps on sending

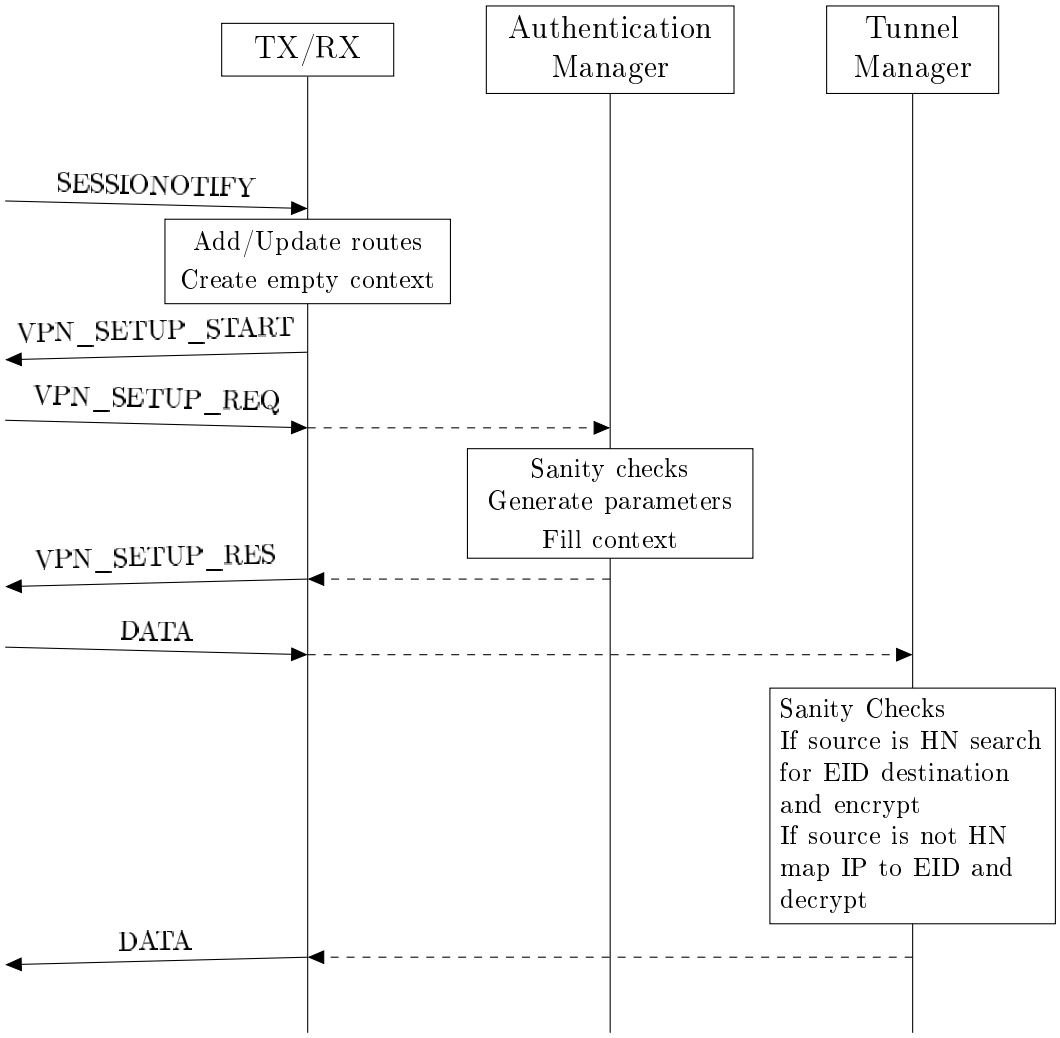



Figure 3.13: Server Packet Flows

through the L2 TX/RX modules. The modules and their relations on vehicles is shown in Figure 3.16. In the figure, the tunnel manager sends or receives data from users, and the authentication manager interacts with authentication messages. The control plane forwarding manager establishes routes in accordance with the messages received, and the data plane forwarding manager uses the structures generated to route packets in the meshed network. The table manager is also required to clean tables when routes are detected to be invalid (when they expire).

An example of the flow of packets and operation taken by the OBU with control packets can be seen on Figure 3.17, and a diagram of the operations taken on data packets can be seen on Figure 3.18. As shown in the figure of the control flow, the control plane forwarding

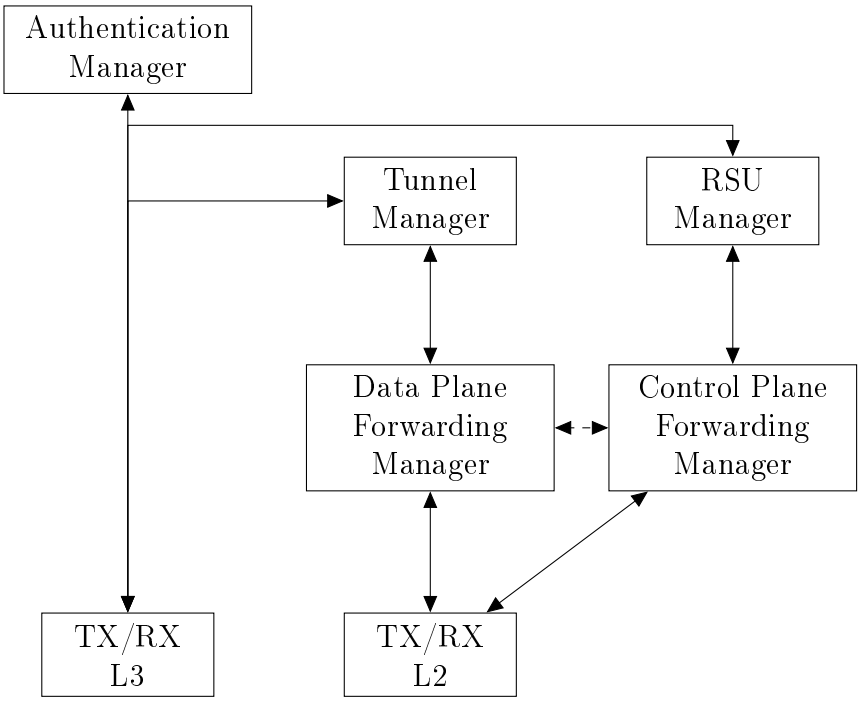



Figure 3.14: RSU Components

manager reacts to packets received from the network in order to construct the tables used in the meshed forwarding scheme. In the figure showing data flow, when a packet is received, the destination is always checked, and if the current node is the destination, the authentication packets will go to the authentication manager and the user data will go to the tunnel manager. If the packets do not belong to the node, the data plane forwarding manager will check the tables and route the packet.

### 3.10.1 Queue Management

Since the queues between modules must not be allowed to fill the memory completely, a maximum size is defined for each queue, and a queueing policy similar to Random early detection (RED) [12] is implemented in order to prevent memory from filling too fast. The algorithm calculates the probability of rejection of the packet based on the current queue size, and it is defined as shown in Equation 3.2. This leads to a rejection probability

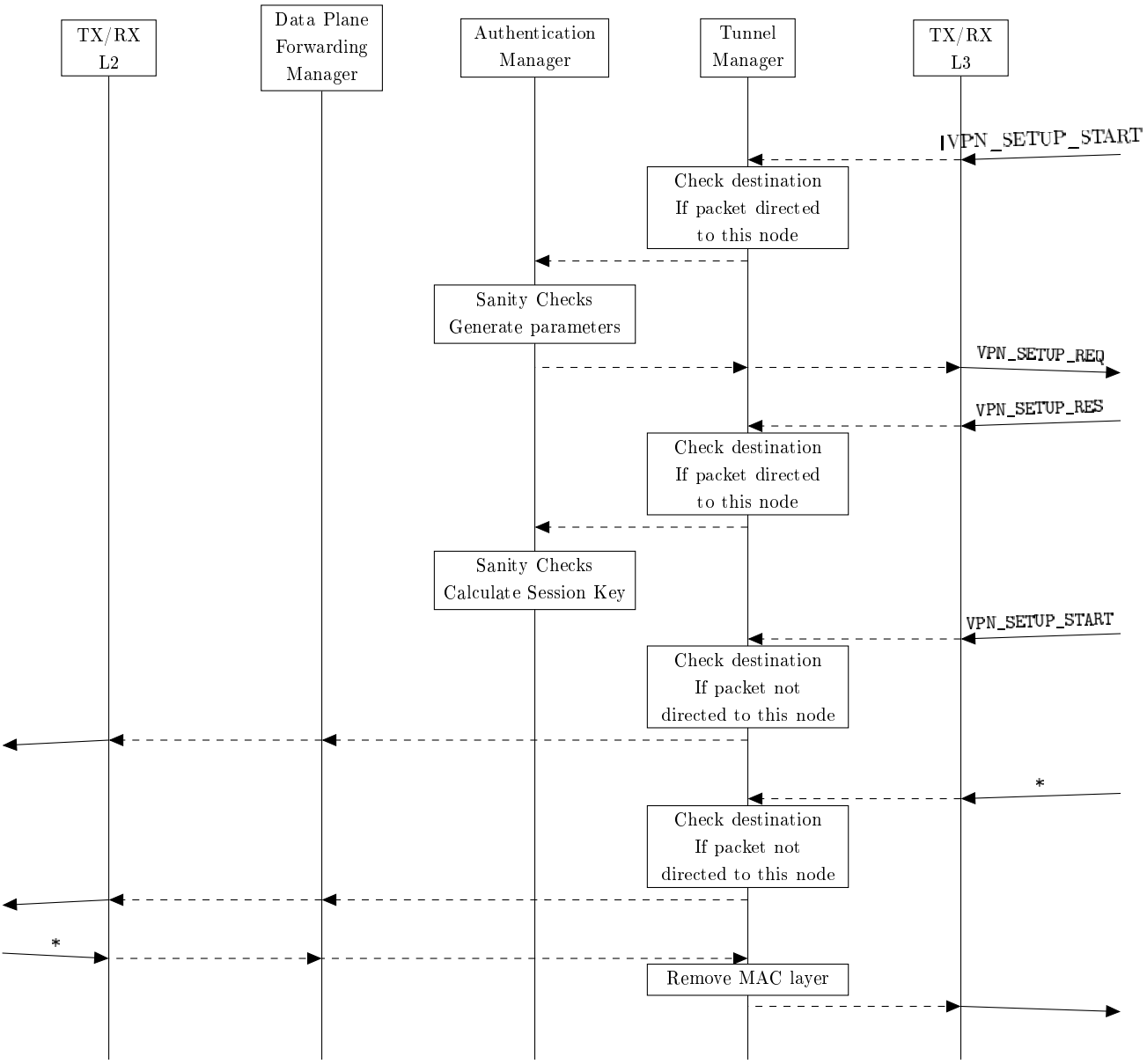



Figure 3.15: RSU Data Packet Flows

function that behaves like described in Figure 3.19.

$$P(n) = \begin{cases} 0, & \text{if } \frac{n}{MAXSIZE} < THRESHOLD \\ \frac{n}{MAXSIZE} - THRESHOLD, & \text{if } \frac{n}{MAXSIZE} < 1 \\ 1, & \frac{n}{MAXSIZE} = 1 \end{cases} \tag{3.2}$$

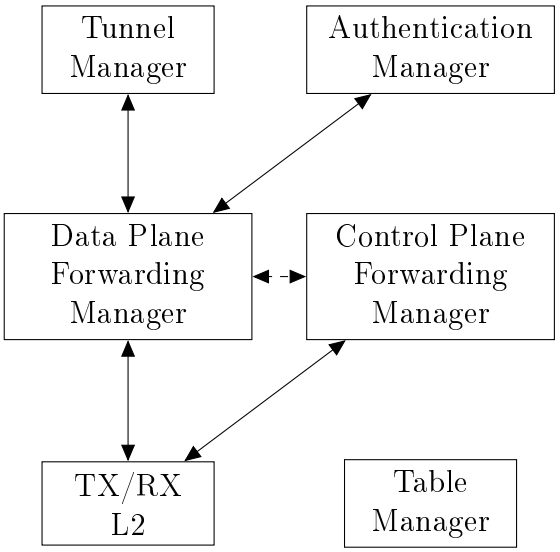



Figure 3.16: OBU Components

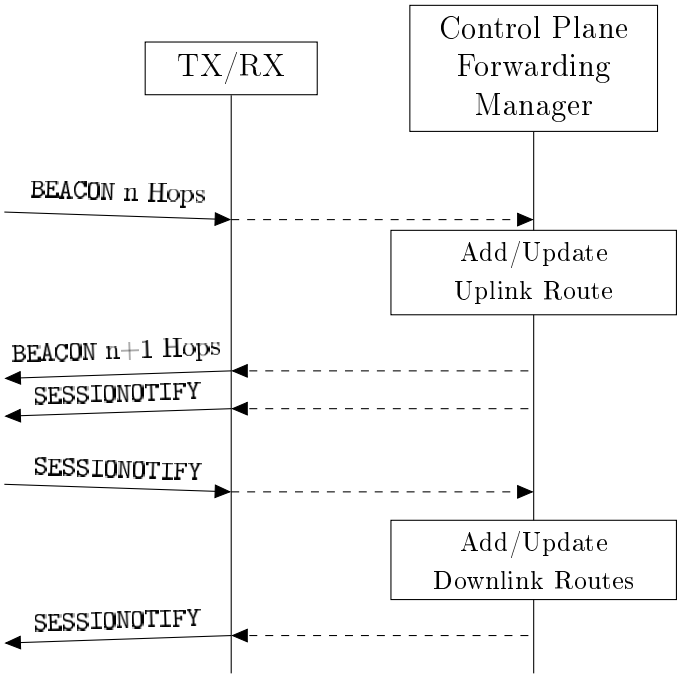



Figure 3.17: OBU Control Packet Flows

---

This allows the memory to be controlled and it prevents the program from being killed by the kernel for using excessive amounts of memory.

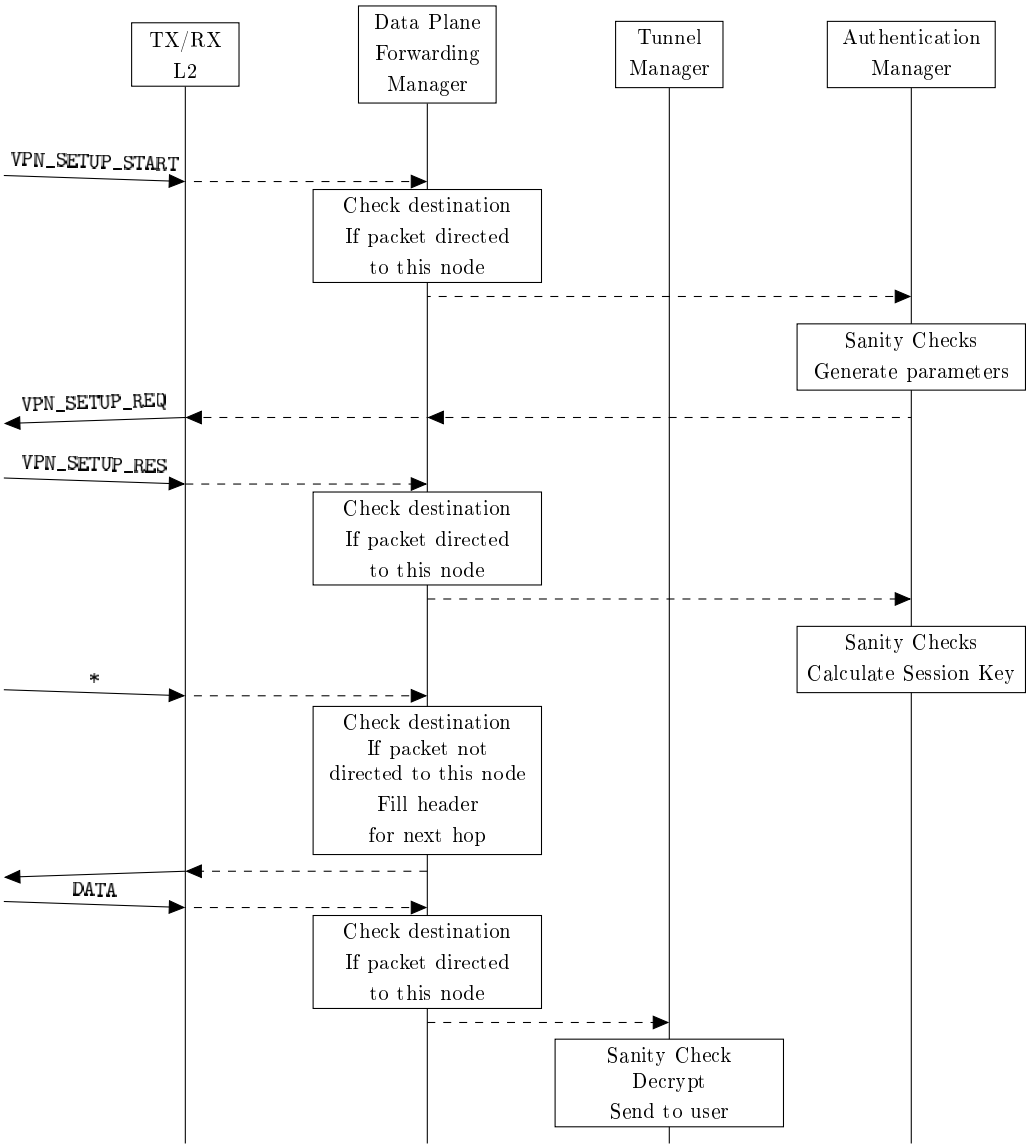



Figure 3.18: OBU Data Packet Flows

## 3.11 Anticipated problems and solutions

### 3.11.1 IV randomization

The IV is a buffer used by the AES-GCM algorithm with the purpose of randomizing the data packet. Using a constant IV in several packets means that, given a certain packet, if there is an equal one in the same session, the result of the cipher will be the same.

This is not a very concerning matter. Even the standard points out that using unique and random IVs is not mandatory. It is, however, recommended, because it might help to

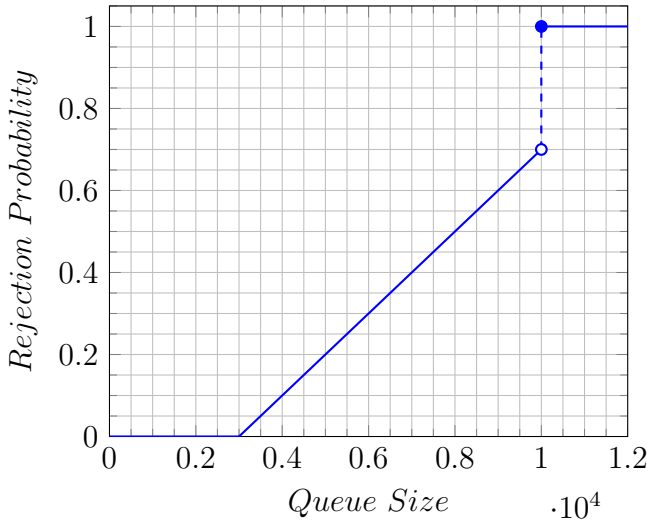



Figure 3.19: Queueing policy implemented

prevent replay attacks. If we maintain the IV equal to a part of the key, we do not need to carry it because both endpoints have that information fixed (using a specific part of the session key, which both endpoints possess).

### 3.11.2 Forwarding

The routing mechanism implemented is a insecure protocol. An attacker could mislead data traffic by advertising beacons with low metric.

This issue may be solved, for example, by appending a digital signature to routing packets. However, unilateral authentication would not be enough, in order to avoid Man-in-the-Middle (MitM) attacks, mutual authentication would be required which would probably have impact in the performance of the routing algorithm.

Another way this could be solved is to implement a hash chain [56], because hashes have the advantage of being very fast to generate. A disadvantage of this is that this is only good for $N$ passwords where $N$ is chosen at initialization time and a counter of passwords must be maintained [41].

### 3.11.3 Scalability with an increasing number of clients

In order to allow inter-node communication, NAT deployment must be done in the server and not in the OBU. NAT deployment, if done at the OBU level, will allow the server to route traffic to nodes just by knowing the IP of the vehicle.

However, if NAT is done at a vehicle level, when intercommunicating between clients, the server will not know where the destination IP of the client is located. This cannot be solved easily, since a NAT deployment in the OBU level will increase the solution scalability, but would deny inter-client communication. A NAT deployment at the server level would allow this feature, but would decrease the performance, since each user has to be mapped instead of only its network access OBU.

### 3.11.4 Link Layer Broadcast Flooding

Most operating systems provide virtual devices, the `TUN` (network tunnel) or the `TAP` (network tap), that are not attached to network adapters. They are usually used in tunnelling solutions and virtual-machine networking. The `TUN` device is used in IP-only applications and the `TAP` operates with Ethernet layer. In tunnelling solutions, `TAP` implements a bridged network and the `TUN` device can implement a Point-to-Point IP-only network.

Using a `TAP` device to capture users' traffic, layer 2 broadcasts are caught in the device. This would cause broadcast flooding (mostly from neighbour discovering services) because, when the server received such packets, it did not know how to reply, and because of that, it would replicate that packet for each board on the network.

In order to solve this, which is a known problem of bridged VPN [52], the solution is to separate broadcast domains by changing the device to a `TUN`, which is Point-to-Point IP-only device.

## 3.12 Chapter Considerations

The current chapter presented the proposed architecture. The solution uses known cryptographic primitives to secure communication between a node, and a external network, supplied by a service provider.

This tunnel is secure, because the stakeholders exchange a symmetric key in a mutual-authenticated DH exchange. The key derived from this protocol, is used not only to assure the privacy of the data flowing in the network, but also the integrity of that same data.

The network is, at the time, vulnerable to attacks on the routing mechanism. These attacks can create partitions on the network. They cannot, however, impersonate the stakeholders in the authentication process, and neither can they interpret the data sent by the user or the server, once authentication is completed.

# Chapter 4

# Secure VANET implementation

## 4.1 Introduction

This chapter describes the implementation of the proposed solution.

In section 4.2 it is described the paradigm which fits best the solution, and the libraries used. Some libraries have advantages in comparison with others and, because of that, this is crucial information that describes why those decisions are made.

In section 4.3 structures used to hold information about routes, cryptographic parameters and user info are described.

In section 4.4 are described the functions which allow to use the security protocols.

In section 4.5 it is described how the traffic is made confidential after authentication.

In section 4.6 it is described, the decisions made by modules within each node.

In section 4.7 it is calculated the overhead put on the network by the authentication and the components which are holding back the software performance.

In section 4.8 it is described how each node should be configured in order to start this solution.

In section 4.9 the considerations about the implementation are described.

## 4.2 Software Paradigm and Libraries

The software was designed using a Event-driven architecture (EDA) approach so that the flow of the program is determined by network events or local variables which are set at the beginning of the program.

This type of approach may be used by applications which transmit events/messages among loosely coupled software components (components who have little or no knowledge of other components and react only to events occurring in themselves). It is also a good strategy because each module which forms the software is asleep when it does not have events queued, saving a lot of processing power when idle.

Because of this kind of detachment between components allowing completely asynchronous operation, several workflows may have to be defined to describe the operation of the whole software.

However, since events are mostly asynchronous, not only deadlocks are bound to disappear but also the software may be able to process two events at the same time in different components, when synchronization between data is not necessary.

Several libraries are used in order to provide easy access to structures and operations that are required in the flow of programming.

In order to have at disposal several data structures which are going to be used along the development, there is a need to choose a library which supplies access to data structures. In `C++` the Standard Template Library (STL) has an efficient implementation of generic data structures which would be a major advantage of using this language. In `C`, however, most libraries use `void *` which is slow to instantiate, may cause cache misses and is not optimized by the compiler. The other way of obtaining a generic library in `C` would be to use macro defined structures in order to instantiate every structure at compilation time instead of runtime. By doing this, we can have solutions as efficient as type-specific libraries, which are able to work with custom-defined structures. The only drawbacks would be that macros may turn code hard to debug, less comprehensible and the dynamically allocated memory handling inside each node must be handled by the developer and not the library.

To handle data structures, `KLib` is the library chosen. It provides several data structures which do not need to be installed, and each one has no internal or external dependencies. The data structures included in the project are `khash` and `klist`.

`KHash` provides a hashed list implementation and is used to create structures when lookup operations are common, because hash lists are the most efficient structures in these operations.

`KList` is the queue implementation used to pass data between threads, and also used to handle routing tables. The reason why queueing is chosen to be implemented as a FIFO is straightforward. Linked-Lists are very efficient when appending data in the tail and removing the head, each one of these operations needs only one command. It is not

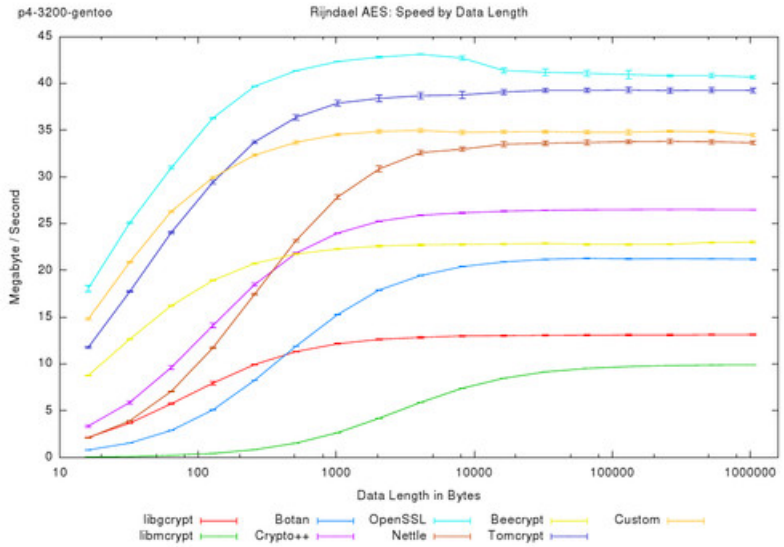



Figure 4.1: Performance comparison between several cryptographic suites tested in a computer with a Intel Pentium 4 at 3.2 GHz with 1024 KB L2 cache [4]

---

efficient, however, to iterate over data because no shortcuts are available and the list has to iterate over each and every element until it finds the one needed.

Yet, even with the reasons presented earlier, routing tables do not gain advantage over using neither one of the two implementations refereed. The reason why they are kept with linked lists is because the algorithm might change, thus changing the search key. This would imply that the Hash table implemented now would not be valid in other algorithms and had to be rewritten. By keeping them implemented as linked-lists, the only way to search for nodes is iteration, and while this is a not performance advantage, it would help to maintain compatibility with big changes in structures if the lookup ID is maintained unique, and of course, linked-lists may be sorted whilst hashed lists may not guarantee this.

For cryptographic operations, OpenSSL is used because it is the suite of low-level algorithms most optimized as show in Figure 4.1, which also requires the use of `libcrypto`.

OpenSSL provides low-level interfaces with `BIGNUM` structures used to store keys and make arithmetic operations on numbers with uncommon sizes, ciphering algorithms and hash functions.

The modules are not sub-routines, they are co-routines. This allows multiple entry points and are suited to state-machines and communication between modules.

For the module implementation, the library used to create threads is POSIX Threads,

known as `pthread`. Pthread is a library of high-level functions, types and constants which simplify the creation and management of userspace threads. It is probably not the most efficient implementation to use. The most efficient would probably be `clone` with the support of `futex` calls. Other libraries are also available, for example, GNU Portable Threads or even State Threads. However, as `pthread` has much more documentation and use case examples, it seemed simpler to just use it.

It was necessary to decide also how to implement the modules described in section 3.10. The options would be processes or threads. Processes had the immediate disadvantage of separating the memory of each module. This problem could be solved in two ways:

- Create a big chunk of memory, shared between every module and a fixed zone to index it.
- Create a system to manage dynamic, shared memory. Since shared memory is not implemented dynamically, this would imply creating a memory zone just for indexing and then access those indexes in different shared memories (in the same way a file-system has a indexing zone and a separated data zone).
- Create a protocol with UNIX sockets or pipes to pass the data structures between processes.

The first two options would complicate too much the synchronization process, and the third option would take a lot of effort to implement, because every operation on structures have to be performed in an internal protocol (but it would solve concurrency problems). However, the better option seemed to be the implementation with threads whilst passing data within the shared memory.

`Pthread` library allows to use the same memory stack provided by the kernel to be accessed to all instantiated co-routines. This memory must be protected with atomic structures supplied by the library such as semaphores or mutexes. The chosen structures were mutexes which are basically binary semaphores without signalling. The reason for this is that, whilst semaphores fit the producer-consumer paradigm, some threads were not meant to be eternally asleep if there was no producer. This was a simple task to implement using mutexes and condition variables with timers, which allows not only to sleep until a condition signal is received, but also wake up after a certain time.

To pass data between threads, there are also several options. The easiest one is to use UNIX Sockets, which allow processes and threads to communicate as if they were network

devices. However, the current implementation which uses a FIFO in a shared memory uses one copy between most threads and can be adapted to use zero copies. UNIX Sockets would forcefully use two copies because, on send operation packet would be copied to kernelspace, and then, on the receiver the packet would be copied again from the kernelspace to the userspace. Pipes (or named-pipes) can also have been used and they would allow to pass data between threads through the kernel, however named pipes are more useful for one-to-one communication while shared memory can easily support many-to-many.

## 4.3 Software Structures

Several structures are created in order to make the information handling efficient.

Data is exchanged between threads using linked-lists because they are efficient to be used as FIFO queues. Elements are always queued in the last position using the tail element, and always removed from the head element. Both operations' complexity is $\mathcal{O}(1)$ because first and last positions are known.

This data structure, show in Figure 4.2, contains a pointer to char where packets may be allocated, a size variable and a source variable, which allows threads to confirm the thread where the packet originated, and deal with that information in accordance.

The library deals with the memory from the nodes, because it has a memory pool where memory is allocated and managed. However, the implementation is made without assumptions of the internal elements allocated inside the structure and, because of that, the memory allocated for each packet must be freed.

When a module needs to send data to another one, it locks the queue owned by the destination thread, fills the source, the destination and the packet size field, and then allocates memory inside that structure in order to copy that packet. Each thread is waiting for packets to fill their queues, so when a packet is received, in order to reduce the time locked, the thread gets all the data it needs from the packet, frees the packet memory and unlocks the queue. This way, the time used to allocate the packet in a thread will overlap, as seen in Figure 4.3, with the data processing on that module, because both operations are parallel, making the time of generating copies less restrictive.

Routing structures are kept as linked lists, because they are easier to deal compared with other structures, and offer a way to iterate, which does not need the knowledge of the items inside, because of that, the complexity to search is $\mathcal{O}(n)$.

The hash structures, shown in Figure 4.4, have as a basis the `KLib` implementation,

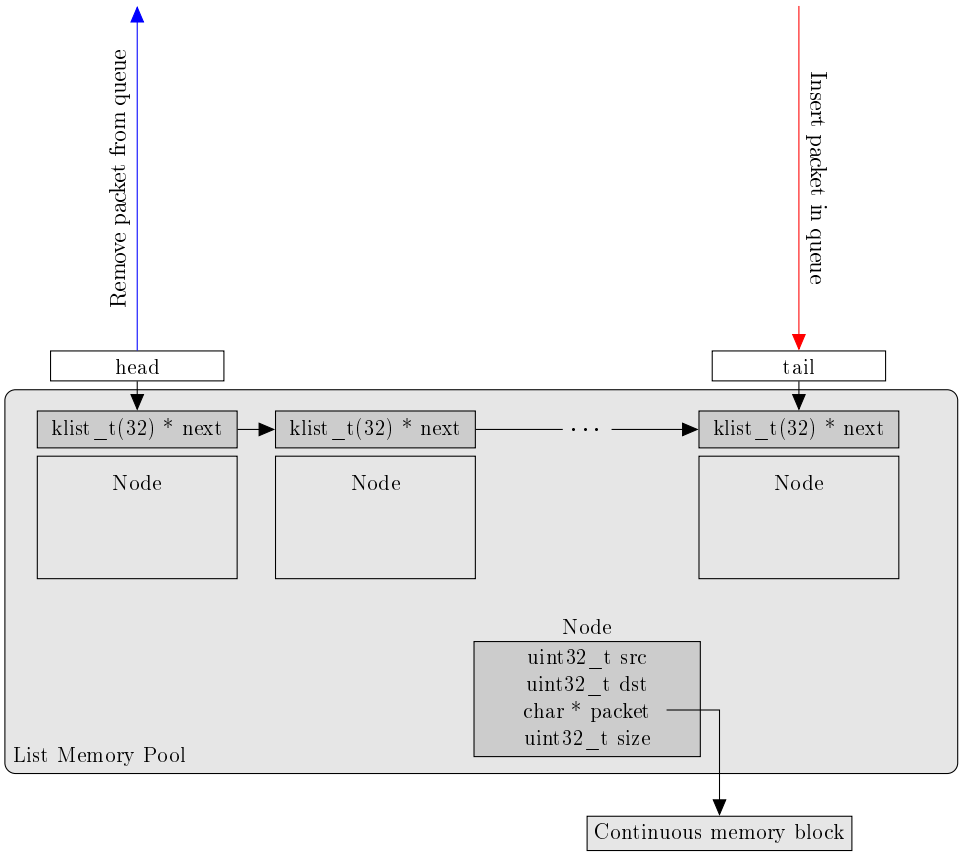



Figure 4.2: Internal Queue Structure

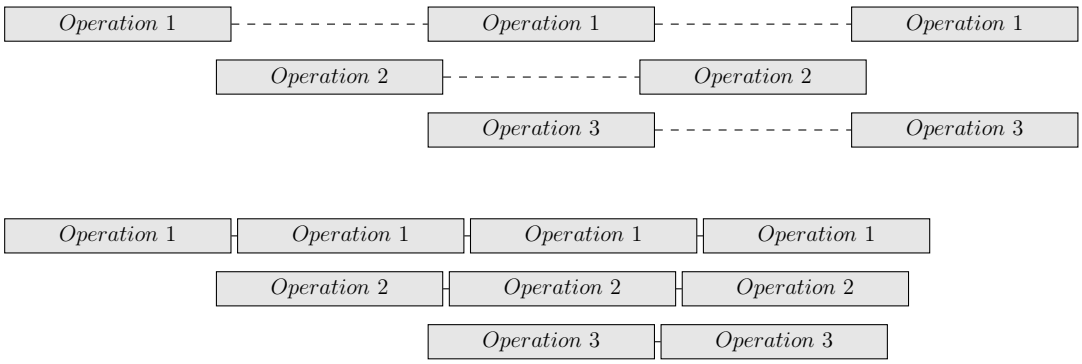



Figure 4.3: Example of multiprocessing strategies. On top each thread is blocked while other threads processes some piece of data, while the approach taken in the figure below has the advantage of reducing the gaps and take advantage of every cycle the thread can get from the processor. The gaps represent places where threads are blocked in order to put shared data available

---

which uses open-addressing double-hashing to solve collisions. The main idea is that using this, the index in the table where a Key is going to be stored is $Index_1 = Hash_1(Key) =$

$Key \mod Table_{Size}$ and, if a collision occurs, the position should be recalculated to find the difference of positions between the first and the second hash [54].

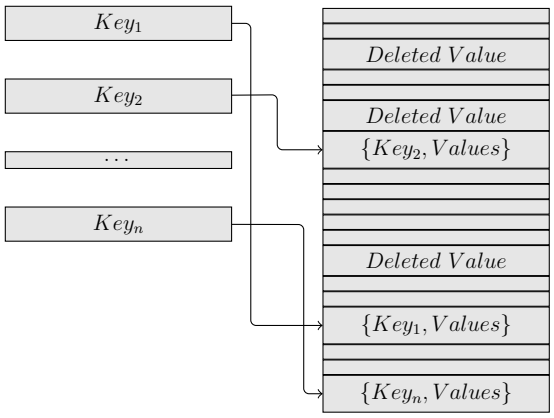



Figure 4.4: Hash Structure

Depending on the purpose in which the Hash table is used, the structure of the values and the key itself might vary.

In the VPN server, the communication is made at Layer 3, and the solution needs to keep track of the Port and the IP belonging to the RSU where a Session is communicating. This allows the server to change a Session downlink route seamlessly. This is mapped using a hash table where the Key is the session, and its content is a structure that holds the current IP and Port of the RSU. The complexity of the access of a known Key in a Hash Table is $\mathcal{O}(1)$ and, because of that the access is very efficient. This IP and Port fields belong to the RSU which is the connection to the OBU advertising this.

The VPN server also holds a structure which informs in which Session a user IP is communicating. This is also a Hash Table and, therefore, it has the same complexity of the structure that maps $Session \rightarrow [IP, Port]$ and this one maps $UserIP \rightarrow [Session]$, allowing the server to direct incoming traffic to a specific user instead of sending to every session available.

The other structure is common to the VPN server, the RSU and the OBU. It is a Hash Table which maps $Session \rightarrow [CryptoParameters, Session\ State]$. By doing this, each OBU/RSU can have its own entry in this table, which allocates the current context, and the server uses this to store all the Sessions contexts.

## 4.4 Security related functions

In order to be able to use security primitives the following functions were made available by libcrypto and OpenSSL:

- `RAND_seed(const void *buf,int num)`: Establishes a seed for the random number generator. The buffer should be random bytes with `num` bytes.
- `BIGNUM * BN_new()`: Allocates and initializes a BIGNUM structure and returns its memory address.
- `BIGNUM * get_rfc3526_prime_2048(BIGNUM *)`: Returns a pointer to a structure containing the parameters recommended to be used in DH exchanges by [30], changing this function will change the key size and the group used.
- `BN_set_word(BIGNUM *, unsigned int )`: This function is used to substitute a "small" number in a BIGNUM structure. Since the generator is always 2 in every group, this function is used to put that value in the structure.
- `BIGNUM * BN_value_one()`: this function returns the value 1 and is particularly useful to do subtractions.
- `BN_sub(BIGNUM * a,BIGNUM * b,BIGNUM * c)`: this function calculates $a = b - c$.
- `BN_CTX_new()`: Allocates and initializes a temporary number used to support calculations. OpenSSL requires this temporary on the functions where multiple operations are made.
- `BN_rand_range(BIGNUM * k, BIGNUM * range)`: Generates a random number from $[0, range[$. It is used to generate random private keys for the DH exchange.
- `BN_free`: Deallocates a previously used BIGNUM
- `BN_add(BIGNUM * a,BIGNUM * b,BIGNUM * c)`: this function calculates $a = b + c$.
- `BN_CTX_start(BN_CTX *)`: Renews a already allocated temporary number.
- `BN_CTX_end(BN_CTX *)`: Finishes the use of a renewed temporary number.
- `BN_CTX_free(BN_CTX *)`: Frees a temporary number.

- `BN_mod_exp(BIGNUM *r, const BIGNUM *a, const BIGNUM *p, const BIGNUM *m,BN_CTX *ctx)`: Calculates $r = a^p \mod m$. This is one of the most used functions used in both asymmetric cryptography and DH key exchange. In the context of the DH protocol, it is used to derive the Public Key from the random Private Key. In RSA, for example, it is used to sign packets and also to encrypt them (depending on what key is used).
- `BN_bn2bin(BIGNUM *, unsigned char *)`: Takes a BIGNUM and copies it into a pre-allocated buffer.
- `RAND_pseudo_bytes(unsigned char * ,int num)`: Puts `num` bytes of pseudo-random data taken from `/dev/urandom` in a buffer. The buffer should already be allocated and the random generator should be seeded.
- `SHA256_Init(SHA256_CTX *)`: Initializes a SHA256 structure.
- `SHA256_Update(SHA256_CTX *, unsigned char *)`: Runs data over the hashing algorithm.
- `SHA256_Final(unsigned char *,SHA256_CTX *)`: Extracts the calculated hash to a buffer. The buffer should have at least SHA256_DIGEST_LENGTH bytes.
- `RSA_new()`: Returns a pointer to an initialized structure prepared to hold RSA parameters.
- `PEM_read_RSAPrivateKey(FILE *, RSA **, pem_password_cb, void *)`: Reads an RSA Private Key stored in PEM format. Using this format, the key may be protected with a password. This password can be hardcoded in the pem_password_cb or a callback could be defined in order to prompt for the password.
- `RSA_free(RSA *)`: Frees a RSA structure.
- `PEM_read_RSA_PUBKEY(FILE *, RSA **, pem_password_cb, void *)`: This function does the same as the Private, but it reads a public key instead of a private. They are different because this one does not read private keys to the structure (because public files do not contain them) whilst the other one fills every field on the structure.
- `unsigned int PadData(unsigned char *, unsigned int len, int blksize)`: Given an array with len bytes, applies padding according to CMS to a buffer and returns its new length. The buffer should have memory allocated before the function is called.

- `int EncryptData(unsigned char * in, unsigned int inlen, unsigned char * out, unsigned char * key)`: Takes the in buffer with inlen, applies padding to the buffer and ciphers the buffer using the algorithm defined. In the end, it appends a TAG in the end of the packet and returns the new length of the packet.

- `int DecryptData(unsigned char * in, unsigned int inlen, unsigned char * out, unsigned char * key, int * auth)`: Takes the in buffer with inlen bytes, deciphers the buffer using the algorithm defined, verifies and removes the padding, verifies TAG and returns the new length of the packet if valid (or -1 otherwise). *auth shall be allocated before calling the program (otherwise verification of the TAG will not be done) and will return 1 if packet is authentic and 0 otherwise.

- `EVP_EncryptInit(EVP_CIPHER_CTX * , EVP_CIPHER *, unsigned char * key, unsigned char * iv)`: Sets up cipher context for encryption. The difference between EVP_EncryptInit_ex and EVP_EncryptInit is that, in the latter, the context does not need to be initialized implicitly. EVP_CIPHER should be a pointer to a ciphering function (in this case EVP_aes_256_gcm()).

- `EVP_EncryptUpdate(EVP_CIPHER_CTX * , unsigned char * in, int inl, unsigned char * out, int outl)`: Updates variables like the AAD, or does the ciphering operation on the packet.

- `EVP_EncryptFinal(EVP_CIPHER_CTX * , unsigned char *, int * len)`: Retrieves data from the ciphering engine.

- `EVP_CIPHER_CTX_ctrl(EVP_CIPHER_CTX * , int type, int arg, void *ptr)`: Allows to set additional parameters, like for example, the TAG length.

- `EVP_DecryptInit(EVP_CIPHER_CTX * , EVP_CIPHER *, unsigned char * key, unsigned char * iv)`: Sets up cipher context for decryption. The difference between EVP_DecryptInit_ex and EVP_DecryptInit is that in the latter the context does not need to be initialized implicitly. EVP_CIPHER should be a pointer to a ciphering function (in this case EVP_aes_256_gcm()).

- `EVP_DecryptUpdate(EVP_CIPHER_CTX * , unsigned char * in, int inl, unsigned char * out, int outl)`: Updates variables like the Additional Authenticated Data (AAD), or does the deciphering operation on the packet.

- `EVP_DecryptFinal(EVP_CIPHER_CTX * , unsigned char *, int * len)`: Retrieves data from the ciphering engine and returns 0 in authentication failure or greater than zero on success.

## 4.5 Data exchange

After both endpoints get an unique private session key, the header referred in Figure 3.10 will be appended with a payload padded to multiples of 16 bytes according to PKCS#7 and CMS, and appended with a TAG after encryption which GCM uses to authenticate data after deciphering.

This way, data does not need to be signed, which is a slow operation and can use a method which can authenticate data and protect its privacy at the same time.

While data is exchanged, the network that is in between the endpoints is seamless in terms of security which make the process completely detached from the routing and the state of other nodes.

It is important to notice that the GCM mode output will be the same for a packet that uses the same Key and the same IV. Consequently, GCM mode alone only provides integrity and not true authenticity. Still, it does more than other modes of operation which usually only deal with privacy and rely on other algorithms to control integrity.

## 4.6 Workflow for different types of nodes

Since different types of messages trigger chain reactions between different components which also differ based on the type of node, the present section differentiates every workflow implemented for each type of message and each node.

### 4.6.1 VPN server workflow description

The first component is a module which receives and sends any type of packets directed at the VPN server address. This thread decides which type of packet arrived at the server and, if identified, queues the packet in another component.

If the packet is of type `VPN_SETUP_REQ`, it will send the packet to the authentication manager module.

If the packet is of type `DATA`, it will send the packet to the tunnel module.

If the packet is of type `SESSIONOTIFY`, the RX Module will set the context for each session that is created. This context includes the EID of the board, the RSU from which it received this information, and the Cryptographic parameters which in this state are initialized to invalid values in order to not be accepted. If the session received in this module is a session which is not known, the server will send a `VPN_SETUP_START` to the node in order to request data from the node. The purpose of this module is also to keep updating the RSU which serves as gateway to a certain EID.

In the authentication manager module, the context, which was created in the session manager module (if it was not, it will be created here), will be filled more in depth. Since the only packets accepted by this thread in the server are `VPN_SETUP_REQ`, the packet already contains the DH public key and the Signature header.

In this module, the Signature header will be checked, in order to know if the packet is authentic or if the packet was not modified by some non-authorized third-party. If the packet is marked invalid, the thread will discard the packet and no actions are taken in the context. If the packet is deemed valid, the server will calculate the private session key using its private DH key, and will send his public key to the node so that the node can calculate the Session key, by sending a `VPN_SETUP_RES`.

In the tunnel module packets may have different flows based on the network where they are received. If they are received from the outside network and they have a header with the EID, the packet is going to be deciphered and checked for integrity using the AES-256-GCM implemented in the `DecryptPacket` function. Then, since it is a packet sent by a user in the meshed segment of the network, the program needs to map its IP address linking it with the current EID where the packet came from. The packet is now ready to be sent to the network as the user sent to the board.

If the packet is received from the HN, this module will check if the destination IP address is known in the list, and if it is, it will cipher the packet with the key corresponding to the EID where that user is located, and then will add the forwarding header and send it to the corresponding RSU.

### 4.6.2 RSU workflow description

The TX/RX modules receive data from the network and send data to the network. The difference between the L3 and the L2 is the Layer at which the user-space software can access information. In the L2 module, a `SOCK_RAW` is created allowing the software to read and send data starting at the Link Layer Information filtered by the `ETHERTYPE` field. This

type of sockets allows programs to develop protocols at the link layer. The L3 modules only receive/send data starting from the network layer, which are more suited to develop protocols working over IP, since the lower layers will be filled by the kernel according to its tables.

Similarly to the server, the table manager module updates the shared information stored in linked lists. For the RSU specifically, it maintains the list of all nodes connected to itself as the uplink path to the server. The sessions downlink routes expire after `DTABLEEXP`, and the server uplink path route expire after `UTABLEEXP`. These constants are calculated by the preprocessor based on the time of beacon retransmission (`PROVIDERADVTIME`).

The Control Plane Forwarding Manager is the module which constructs the routing tables. Because each module reacts to events, this thread is fed with information from the RSU Manager modules stating that he has 0 hops to the gateway (because it is itself). By doing that, the RSU will send to the network that the nodes have access to the gateway with 1 hop. This module also receives `SESSIONOTIFY` packets which are sent by nodes connecting to the RSU.

The Data Plane Forwarding Manager uses the tables formed by the control plane module to forward packets in the layer 2 segment of the network. In the case of the RSU, this module passes the packet between the Link Layer network and the Network layer segment by using the appropriate sending and receiving modules.

### 4.6.3 OBU workflow description

The Control Plane Forwarding Manager reacts to `BEACON` packets in the network and connects to the better one using the number of hops and the RSSI as a metric. After connecting to a beacon node, the OBU sends a `BEACON` with a worse metric by one, so that a node which is in range of this node but not in range of the one providing service to this one will use this node as uplink connection to the server, and then it sends a `SESSIONOTIFY` to inform the nodes closest to the server that someone is connecting there.

The Data Plane Forwarding Manager uses the routes established in the Control Plane Manager Module to forward packets between the nodes in a way that can pass between traffic between downlink connections, itself and the server.

The Tunnel Manager Module receives data from the user, encrypts it and sends it to the network and to the data which is received from the Data Plane Manager, decrypts it and sends it to the user.

## 4.7 Network Overhead and Software Performance Analysis

In terms of traffic encrypted in the network, the software pads data received from the user to multiples of 16 bytes, and always adds 16 bytes to get a TAG which, in this cipher mode, provides integrity control.

The overhead included by first time authentication of a node is the size of the Signature header plus two messages that contain a byte with the key size, a DH public key, and a Signature header, in each packet. In terms of traffic that gives us 673 bytes per node entering the network.

The overhead of the authentication on key timeout is 620 bytes per node, because there is no `VPN_SETUP_START` packet.

There is some overhead included by the software design in the network, because (just in data packets) several copies of the packet are made along the processing of the packet. This is hard to solve because user data must be padded and encapsulated.

The reason why this does not apply to control packets is that they are internal to the software and do not need to be encapsulated. Because of that, it is possible to implement the control packets with just one buffer copy, which is the from kernel-space to user-space and vice-versa.

Although it would be advantageous to implement this solution in the kernel-space, because the packet ring buffer is written by hardware, in this proof of concept, it is implemented in the user-space where at least one copy is made. That copy must be made because the kernel required a static allocated buffer, which prevented the software from adding an offset to the packet. Since the packets need to be encapsulated, some padding at the beginning is required. Therefore, the number of copies made along the software tree varies between one and three, depending on which interface is sending the packet.

Currently, AES is implemented in software by the OpenSSL libraries in this architecture. However, in Intel architectures, for example, AES is implemented in hardware [22]. This reduces dramatically the time consumed by the transformation, which can explain the overhead gap between the server and the nodes, using a ARM architecture.

There may be another source of overhead coming from the software implementation, which is the `pthread` implementation with a large context switching between modules. When measured on profiling tools, this problem appeared as the second cause of overhead, and the AES operations being the first one.

## 4.8 Node Configuration

The described architecture implements a VPN over Layer 2, and uses a routing mode which interacts as depicted in Figure 4.5. With the configurations described below, the communication process occurs as follows (for example, an ICMP to address X):

1. The User requests the MAC of the gateway to X.
2. Since TUN acts as a gateway to the HN, it replies the Address Resolution Protocol (ARP) address with it's own MAC address.
3. The software catches the traffic in the TUN interface, applies padding and ciphers the package.
4. The packet is sent in the mesh network along the route specified by the routing algorithm.
5. Reaching the VPN server, the packet is delivered to the software which decrypts the packet and removes the padding based on the information of the vehicle.
6. The packet leaves the HN with the public IP address of the VPN server (if NAT is deployed at the server)
7. X replies and the packet is routed by the HN through the TUN device.
8. The user-space software catches the packet, searches the destination IP (NAT was already reversed), pads and ciphers the packet with the parameters used by the session where the User is located, and the packet is sent encapsulated again through the HN gateway targeting the RSU.
9. Then, the L2 protocol carries the packet to the correct node and the packet is delivered to the user, decrypted.

Every transition between virtual devices and physical devices (except in certain cases, the HN gateway) applies the security rules described in chapter 3 to the packet.

On top of that, the following configurations are made in order to set the server (the executable will look for a file in the same directory called `ini.conf` or it can be specified by sending the absolute path of the file as an argument).

The following syntax should be followed. The character `#` as the first line character, represents a line comment. The word `echo` represents a command which will be run in the

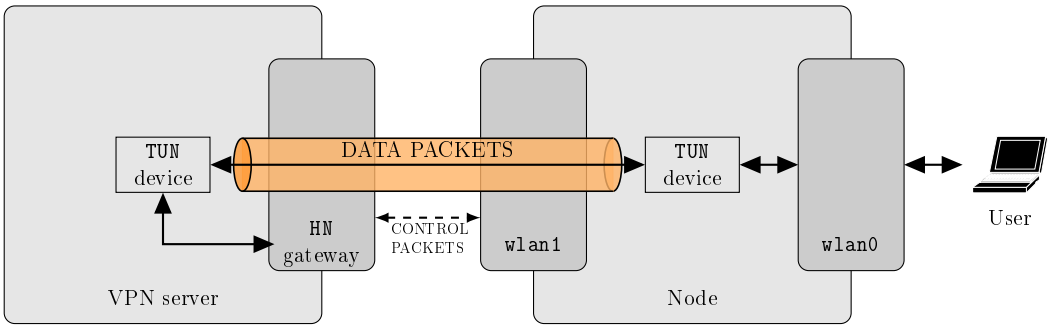



Figure 4.5: Device Setup and Interconnection

system during the software initialization. The word `INTERFACES` is used to specify a list of interfaces to be used to execute the L2 protocol. These interfaces should be separated by a hyphen. On RSU the server IP should be specified by using the keyword `IP`, followed by the IP of the server. In the server, the line `enforce cipher` can be specified to force all nodes of the network to use the cipher algorithm. In the nodes, the default will be to use the cipher algorithm, however if the node does not want to use it and the server is not obliging that, the line `no cipher` can be used to turn off the cipher algorithm.

Also, it is assumed that the IP and the characteristics to the virtual interface are given either in the configuration file or in an external script.

1. The server is configured the following way (this is the ini.conf file):

```
# You should substitute X by the number of device that will be
# allocated. This value is usually 0
echo ip link set dev tunX up

# Give the address to the TUN device. These addresses should
# be out of range of DHCP pools across the network.
# This allows this interface to be used as administration.
echo ip addr add 10.0.0.1/8 dev tun0

# Here you should configure the maximum transmission unit for the
# tunnel. Be aware that since data is going to be encapsulated
# and the MTU for the interface which will carry the encap.
# data is 1500, you should give it way less (1300/1400 is ok)
# Substitute Y
echo ifconfig tun0 mtu Y up
```

```
# Since this VPN on top is made with routing the node has to enable this
echo echo 1 > proc/sys/net/ipv4/ip_forward

# Just to enforce the rules and make sure other rules
# are not valid when the program starts
echo iptables -t nat -F
echo iptables -F

# Deploys NAT in the server. Z is the interface where NAT is deployed
echo iptables -t nat -A POSTROUTING -o Z -j MASQUERADE

# Accept traffic in TUN device (the default should do it but this is just to make sure)
# Again X is the number of the TUN device
echo iptables -A FORWARD -i tunX -j ACCEPT
echo iptables -A INPUT -i tunX -j ACCEPT
echo iptables -A FORWARD -d 255.255.255.255 -j ACCEPT

# Here one should put in W the route for the networks spread by the boards
# (example: 40.0.0.0/8)
echo ip route add W dev tun0
```

2. RSU and OBU can be configured by the exact same script since the type of node and its ID will be read from the files `/root/type` and `/root/id`. The RSU/OBU should be configured this way:

```
INTERFACES wlan1

# Especially RSU nodes should be informed of the IP of the server in order to
# make communication across networks possible. (Ports should be also forwarded
# accordingly)
IP serverIP

# no cipher is used if client doesn't want security
no cipher

echo ip link set dev tun0 up
```

```
echo ip addr add 10.0.6.39/8 dev tun0
echo ifconfig tun0 mtu 1400

echo echo 1 > /proc/sys/net/ipv4/ip_forward
echo route add default gw 10.0.0.1
echo echo "nameserver 8.8.8.8" >  /etc/resolv.conf
echo echo "nameserver 8.8.4.4" >> /etc/resolv.conf

echo iptables -t nat -F
echo iptables -F

echo iptables -A INPUT -i tun0 -j ACCEPT
echo iptables -A FORWARD -i tun0 -j ACCEPT
echo iptables -A FORWARD -d 255.255.255.255 -j ACCEPT
```

3. Apart from this configuration, the nodes should be communicating in the same channel by using:

```
uwme insertChan channel 174 rate 6 txpower 23
```

4. And also in order to have larger bandwidth, Wave Management Entity (WME) stack should be turned on by using:

```
uwme startPS action add psid 80-06 channel 174 access 1 repeat_rate 30
        ip_service priority 12 was_type unsecured psc netname
```

## 4.9 Chapter Considerations

This chapter presents several aspects of the implementation. It explains how software is constructed, which allowed the modules to be separated, presents the libraries used during the development process of the solution. Then, the structures used to store data regarding IP's, Ports, cryptographic material and users, are explained. The workflow of each module implemented in the nodes, is also made and depicted in state diagrams. At last, this chapter ends by presenting the overhead imposed by the protocols used and the software because of its implementation, and the configuration done in each of the different nodes in order to use the actual software.

Since there are various libraries which might do the same operations with different implementation, this chapter compares different sets of libraries based on factors such as, performance or ease of use, to justify the use of one of these libraries. The same is applied to the structures.

In the next chapter, the current solution, is evaluated in different scenarios, to test the efficiency of the software in several environments, and its impact on the network.

# Chapter 5

# Evaluation

## 5.1 Introduction

Network and machine performance is a important factor when considering the tests realized, because they usually involve large amounts of traffic. Hardware used to realize the tests is specified in section 5.2.

In section 5.3, the testbeds used to test the architecture developed, is described along with its several configurations, node displacements, networks characteristics and their implications.

In section 5.4, the most interesting metrics, which can be measured with the setups used, are described.

At last, in section 5.5 we present the results taken on the several setups measured, and compared them, and take conclusions about what is happening to the network.

## 5.2 Hardware Used

The hardware present in the tests, are devices used as the RSUs and the OBUs, and a laptop which was used to simulate the VPN Server.

The laptop has a i7-2670QM CPU @ 2.20 GHz processor which has 4 cores and 2 threads in each core (hyper threading), 3 cache layers of sizes 32 KB, 256 KB and 6 MB, respectively, and supports several hardware extensions (among them the AES instructions for Intel [22]). It has two memory slots occupied with 4 GB plus 2 GB both DDR3 working at 1333 MHz. The disk has a capacity of 500 GB, with a speed of 5400 RPM and 8 MB Cache. The network interface used for tests is a Gigabit Ethernet card.

Table 5.1: Testbed characteristics

| | Server | RSU | OBU |
|---|---|---|---|
| CPU [GHz] | 2.2 (8 cores) | 0.5 (1 core) | 0.5 (1 core) |
| Memory [GB] | 6 | 0.059 | 0.059 |
| OS | Ubuntu (v14.04) | Veniam (v19.2) | Veniam (v19.2) |
| Linux Kernel | 3.13.0 | 3.7.4 | 3.7.4 |

Devices used in the VANET, have a MIPS 24KC v7.4 processor with one core, two data caches with 64 Kb each, and no AES extensions. The amount of memory available, is 64 MB and 128 MB of disk.

Tests were also conditioned by network conditions. In the laboratory, the RSUs were connected by Ethernet to a switch, which limited the traffic to 100 Mbps and the wireless was set to support 27 Mbps which is the current maximum. The tests in the real network required to be connected to a VPN to the core network, where RSUs are connected to. However, the server is in IT network and not their core network. The meshed network is also limited to 6 Mbps.

The server in IT has a AMD Opteron(tm) Processor 4238 @ 3.3 GHz with two cores and AES extensions. It also has 2 GB of RAM.

## 5.3 TestBed

### 5.3.1 Laboratory Testbed

While the solution was on development stage, it was tested on the laboratory testbed shown in Figure 5.1.

This testbed tried to demonstrate the forward mechanism implemented in handover situations to conclude about what would happen to packets which had their route severed and also that it would be possible to minimize reconfiguration of the VPN configuration when that situation occurred. The situations are going to be repeated without and with security.

The first situation was assembled, to measure the route reconfiguration in the server when changing the infrastructure. We expect to the measure the impact of the route reconfiguration on different kinds of traffic, while using the implemented forwarding mechanism. In the first situation we assembled the VPN Server in the laptop described before, and

then moved the OBU between two nodes, which were fixed infrastructure. The criteria for route change in this situation was purely difference in RSSI.

The second situation was tested to measure changes in the meshed network when the route reconfiguration would be made by the intermediate mobile nodes. To do this, we connected the VPN Server in the laptop described previously, in order to interact with a fixed node, which simulated RSUs. We assembled another node, which was fixed, but not connected to any infrastructure serving as a OBU. Also, another OBU is assembled, but this time, it should be moving out of the RSU's range, and entering it again repeatedly, in order to see how fast was the attachment to better connections and how route timeouts affected traffic.

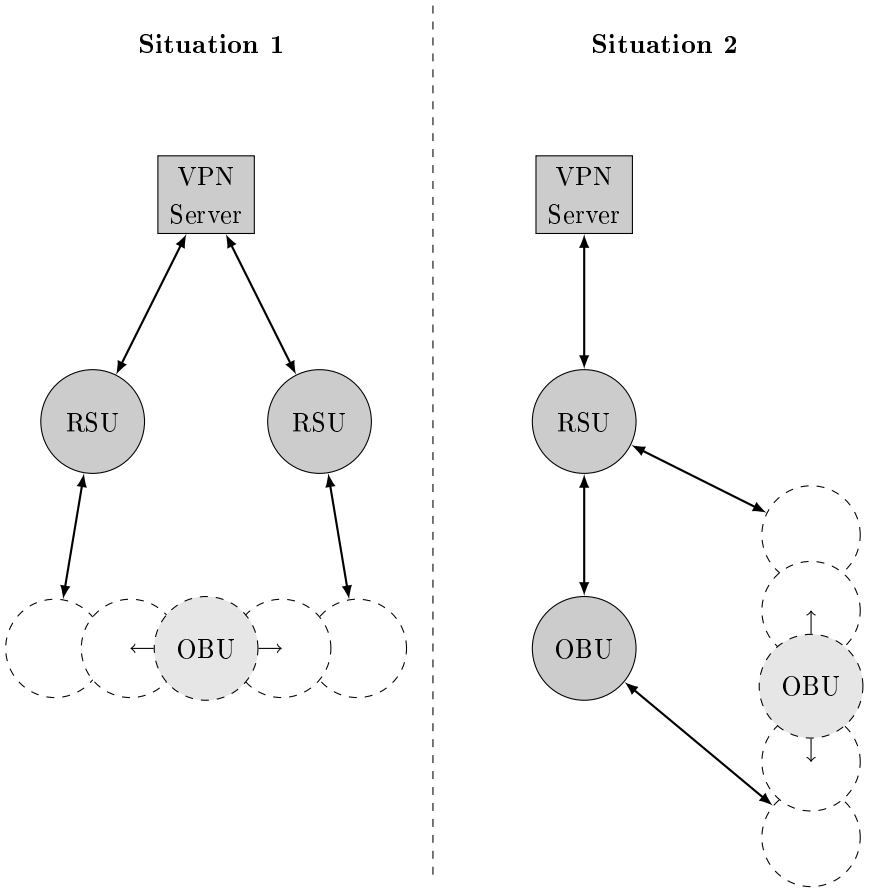



Figure 5.1: Testbed assembled in the Lab. Dashed indicates node movement

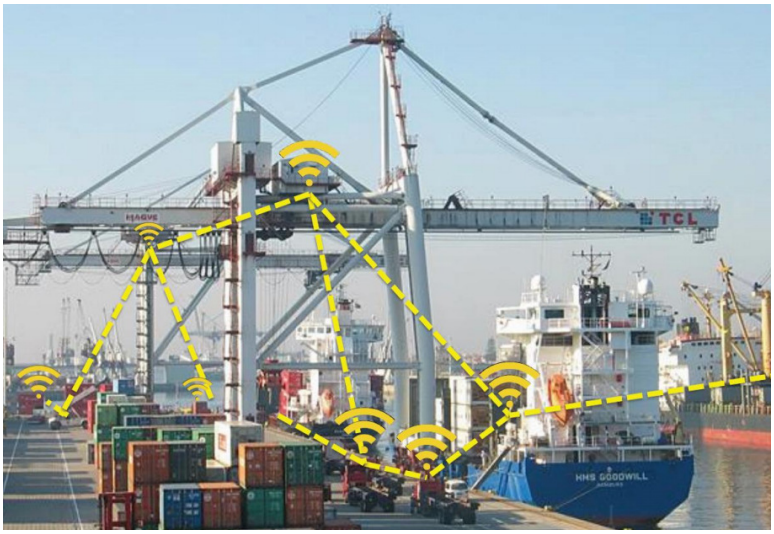



Figure 5.2: Porto de Leixões Testbed

### 5.3.2 Real Testbed

After the tests in the laboratory testbed, it has been measured the same metrics in a real testbed, located in Porto de Leixões, as seen in Figure 5.2. The purpose was to add a lot of entropy to the system, to see what results we could measure from a system with a little more chaos than the laboratory.

Because the testbed is assembled alongside with other architecture serving a real client, we wanted to ensure that service given to them was not affected in any way because of our architecture, therefore, the metrics we were able to obtain were more limited, because we did not have access to the components in place.

We are able to predict some problems in which the developed solution has a lot of challenges. Porto de Leixões still has area which is not covered by the technology, and also some materials present in the local which cause reflections, that may eventually have impact on traffic by creating a lot of duplicate packets. Duplicate packets are compensated by establishing a minimum RSSI for the nodes to establish a connection. By doing that, we can diminish the probability of the nodes causing trouble if the connection is bad. However, range decreases a little.

## 5.4 Metrics

There are several metrics which are interesting to be measured and explained. Since the solution would be used to provide connection for users which will most probably use it for entertainment purposes, the first test realized was the throughput.

The second one should be how much time the user has to wait in renegotiation time until is authenticated in the HN, because, if this time is too large when facing the key lifetime, the Quality of Service would suffer.

The routing was completely decentralized and, because of that, theoretically there should be no blackouts when route reconfiguration happened. Even so, it would be interesting to show the effect on latency and since nodes do not update all at the same time, some packet disordering and loss is to be expected. We wanted to measure the effects of that and take into account the impact in various types of traffic.

In the laboratory, it is also measured the effect of the software on the nodes. This is a very important measurement as the software should not overburden the nodes and therefore impede the normal functionality of other parallel solutions. From this test we concluded throughput tests should be avoided or, at least, reduced to a minimum, in the real scenario, because the network should be operational for another services and, because of that results should be limited in order not to encumber the network.

## 5.5 Measurements

### 5.5.1 Laboratory Testbed

The throughput measured can be seen in Figure 5.3. From this figure, we can conclude that the overhead of the routing, causes the speed of the network to decrease about 8 Mb from the wireless maximum. The difference of throughput between the operation with and without cipher may be explained by the fact that the MIPS architecture in use, does not have the AES instructions processor extension and because of that, takes 0.06 $\mu s/bit$. A model was made in order to explain this reduction. Furthermore, the number of hops did not affect the throughput significantly, and neither did the route reconfiguration. This was to be expected, especially from a small network with low amount of entropy.

The program processes packets coming from the user sequentially, and because of that

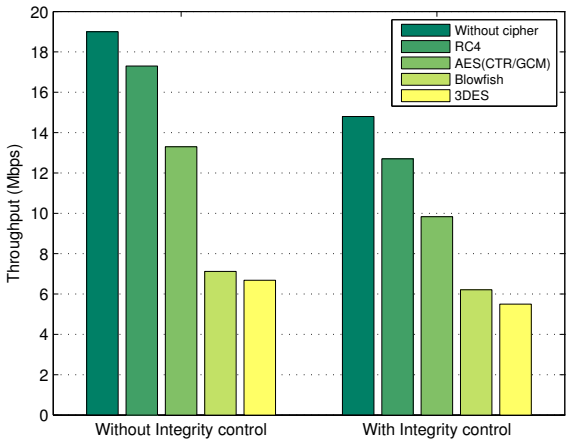



Figure 5.3: Throughput for different ciphers - Laboratory Testbed

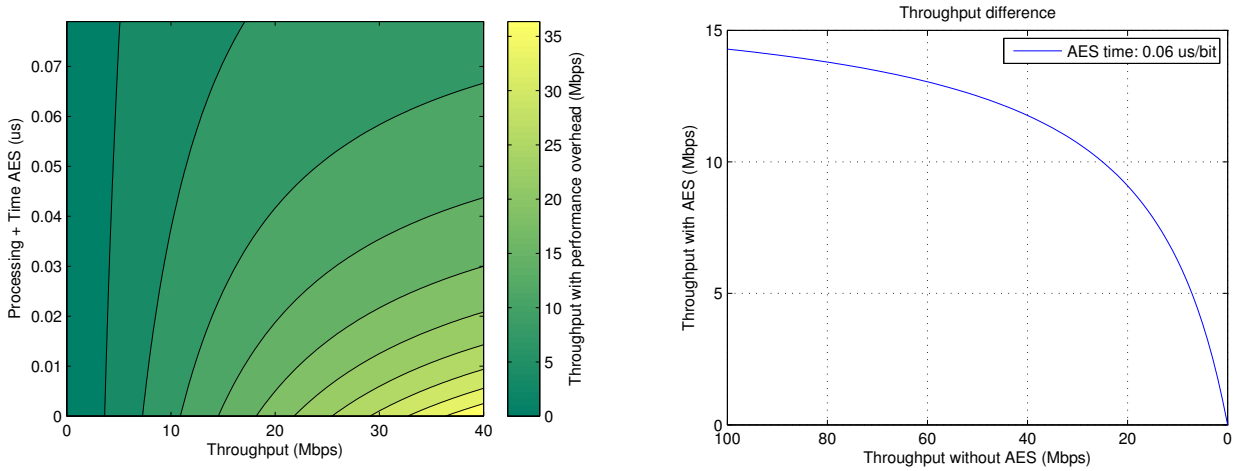



Figure 5.4: AES Empirical Model for the throughput reduction.

the time is added as overhead in all packets, which is described by Equation 5.1.

$$T_{AES} = \frac{1}{\frac{1}{T} + t_{AES}} \tag{5.1}$$

This model, shown in Figure 5.4, was confirmed for several values of throughput, either on a computer which had AES instructions and on the nodes which did not. In the computer with AES instructions the time needed to cipher a bit was $0.001\mu s$ and because of that the limit would be 1Gbps and on the hardware where the software was tested, the measurements indicated $0.06\mu s/bit$, which led to a maximum of 16Mbps. Since the difference between what the software can route in this wireless technology, and in Ethernet (which is already beyond wireless limit) ends up resulting in a small difference in

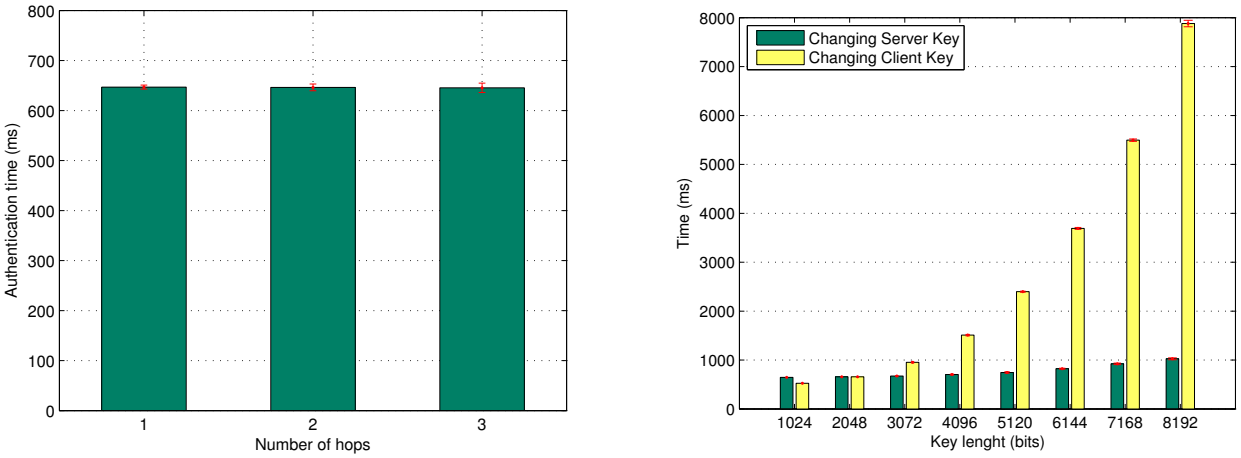



Figure 5.5: The figure on the left shows that the security is routing independent. The figure on the right shows the authentication time while varying the key sizes. When varying one key, the key which authenticates the other endpoint is constant

bandwidth, future improvements should focus on improving some hardware aspects (for example, having multiple cores and processor extensions), until software changes cause a relevant difference.

The second metric evaluated is the time in which each mobile node needed to authenticate in the VPN server. This experiment accounts for the parameter generation and key exchange in both endpoints.

The results from this experiment may be seen in Figure 5.5. In general, this time is irrelevant when compared to the lifetime of the Key generated (even when using the largest key in the server it is about 0.001% of the total time if the key validity is 1 day). The conclusion that was reached was that the function `BN_exp_mod` used in the process of signing the packet and verifying the signature, is the one which adds the delay since it is based on exponentiation of large numbers. Each node should not use very large keys because the time to process the exchange will increase exponentially. The server, however, may use larger keys because the impact is not so large. Using larger keys will also increase the time needed to factorize the key. Since this key is common in every node in the network, it would be advantageous to do this.

The effect of authentication, as seen from the users, were also measured, and it is shown in Figure 5.6. We can conclude that, by shrinking the renegotiation time (to 30s), communication is lost temporarily. However, as soon as the new key is renegotiated, communication from the user restarts to a normal state, seamlessly. Also, keeping the authentication as short as possible, may allow the renegotiation to be almost invisible as

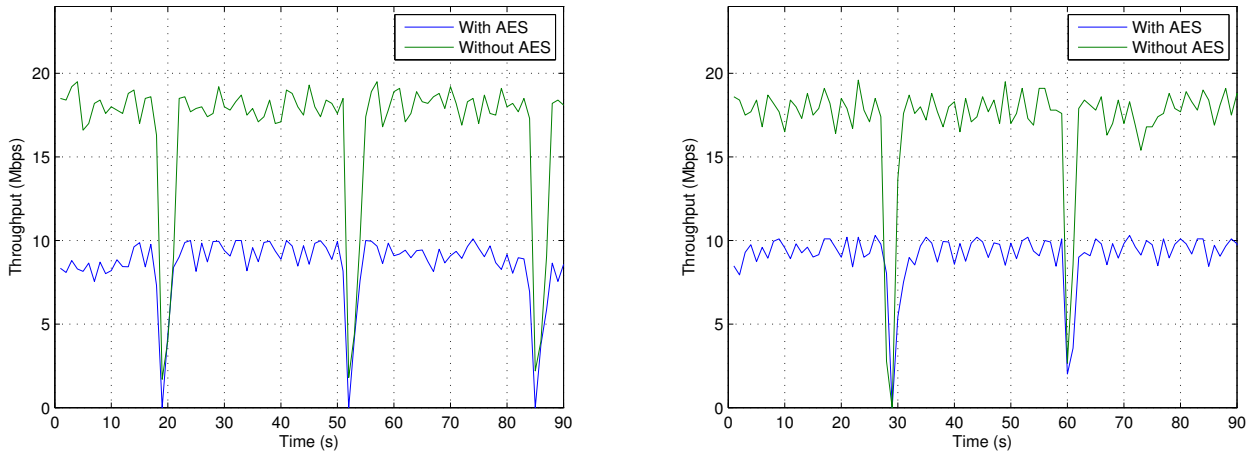



Figure 5.6: Authentication Effect on node throughput - Laboratory Throughput. In the left figure, the server uses a 7680 bit key length, whilst in the right figure, the server uses a 2048 bit key length. The mobile nodes always uses a 2048 because of its lack of computational power.

---

can be seen on the throughput results.

The jitter on the Situation 1 is depicted in Figure 5.7, and it can be noticed a increase of the jitter reported by the used application in some circumstances, but as the software is doing the path choice without ever really disconnecting from the last path (like a soft handover), only a small change is noticeable.

In situation 2, as shown in Figure 5.8, the software has to timeout to connect to the route because the second route connected is worse, and it will not ever connect unless there are no other routes are available. Even so, in the worst case scenario, which would be the last one, the connection is still maintained and the reconfiguration takes little time.

The last metric measured in the laboratory scenario, was packet disordering. This situation was relevant, because as the path changed very fast, and the delay between paths could vary, it was expected to see packets arriving to destination unordered. This is not that important to most services working at application level, because usually, they already have that in mind, and already have a way to fix problems in intermediate networks, as IP does not guarantee reliable packet transmission and its ordering.

It is observed that in the laboratory disordering does not occurs as shown in Figures 5.9 and 5.10, but only a small packet loss. In these figures, the number 0 indicates that sequence number is ordered, positive index indicate packet loss and negative index indicate out-of-order packets.

The disordering problem, may add difficulties to real time traffic, if the disordering

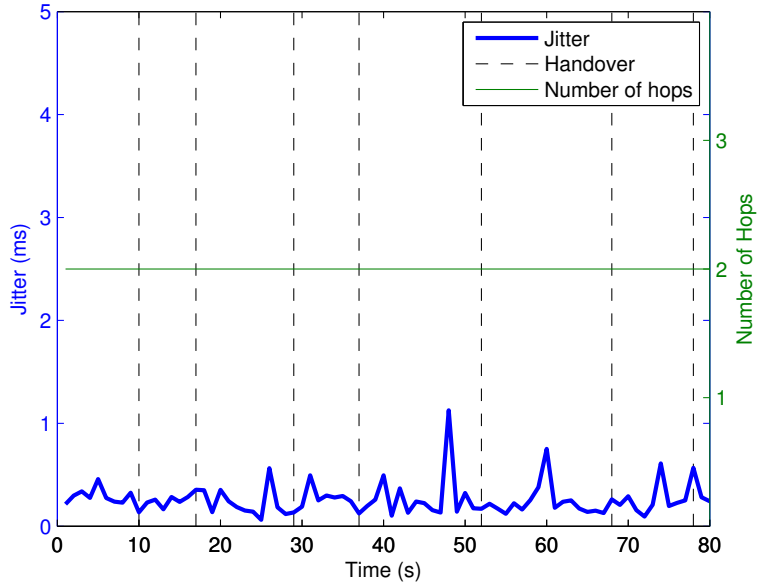



Figure 5.7: Jitter measurement in situation 1 - Laboratory Testbed

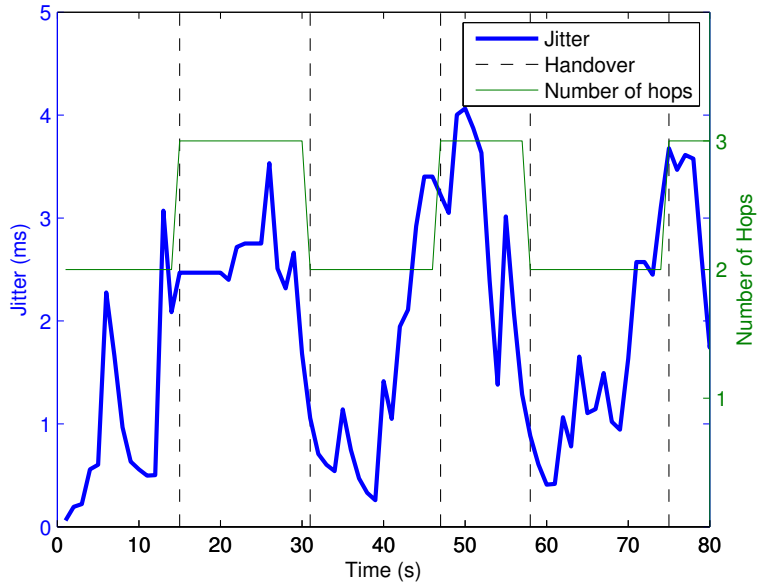



Figure 5.8: Jitter measurement in situation 2 - Laboratory Testbed

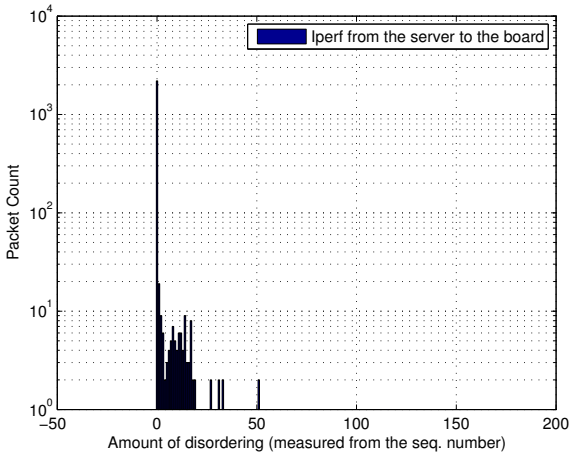



Figure 5.9: Disordering measurement of downstream traffic in situation 1 - Laboratory Testbed

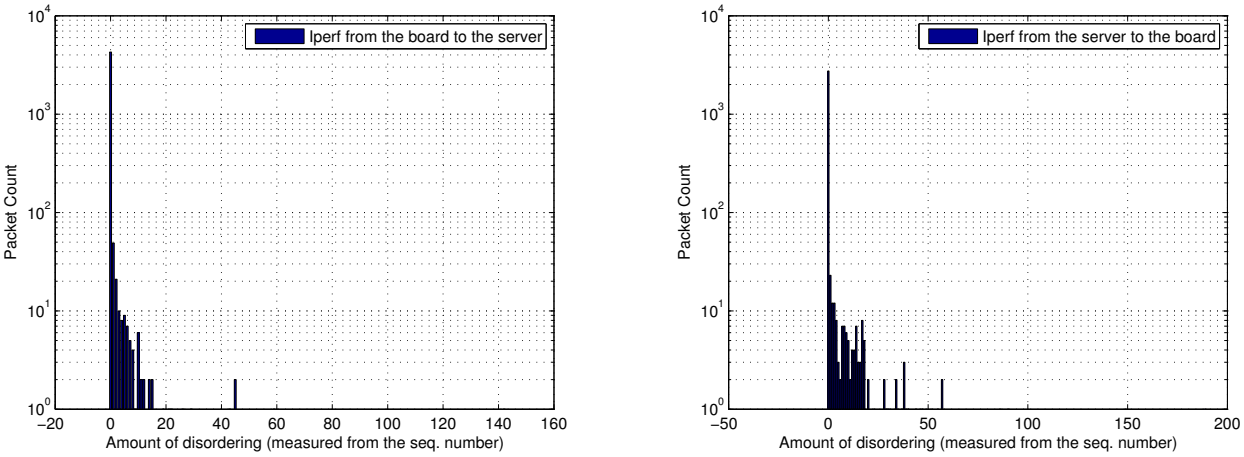



Figure 5.10: Disordering measurement of traffic in situation 2 - Laboratory Testbed

magnitude exceeds the TCP retransmission window. Otherwise, it is not really relevant, and should not modify behaviour of the services used by the client. Furthermore, it was observed that this rarely occurs. It only occurs when traffic has a large bandwidth and the RSSI threshold is removed.

It was also measured the difference of time between one route being used and another route. To do this, a timestamp was kept for each packet along with the MAC carried in that packet. When the MAC changed the client saved the time difference and the next MAC in memory. This timestamp can be interpreted as an handover. The results can be seen on Figure 5.11.

The tests for CPU load and system load, yielded the results shown in Figure 5.12 and

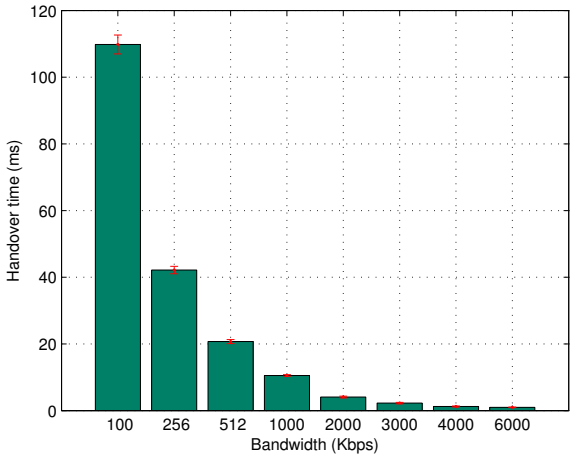



Figure 5.11: Handover Time for different throughputs

Figure 5.13. Memory was also measured and displayed in Figure 5.14. It is important to limit the network, so that, the normal functioning of the software will not have impact on other systems running in parallel.

We could see that, as far as CPU load is concerned, the application does not have that much of a impact, and we can see that it increases linearly as throughput increases, which is to be expected since the processing time increases with the number of packets going through the virtual device.

System load is much harder to explain, since Linux calculates a one minute average of a function of several parameters, that describe system state, such as, network queues congestion, memory and of course, CPU load, by measuring the time processes are sleeping. A good analogy, would be to think of processes and network operations as a bridge. If the number is 0, it means the bridge is empty (no processing to do), if it is 1, the bridge has exactly the capacity which can be processed, and higher than 1 it means there is congestion (processing is slower than the incoming/outgoing flow of packets). It is exactly because of this, that queue management mechanisms were implemented. Since the queues cannot be processed faster that they are filled, in some situations, queues would fill too fast, and the software would be killed by the kernel.

The RED mechanism implemented, solved this question, by discarding packets which were going to obstruct the software. However, logically the load was maintained. To solve this, the software should be able to cope with any throughput inserted in the virtual interface, which means that it is imperative to use zero-copy and faster AES implementations.

It was also measured, the memory used by the software, as seen in Figure 5.14. The

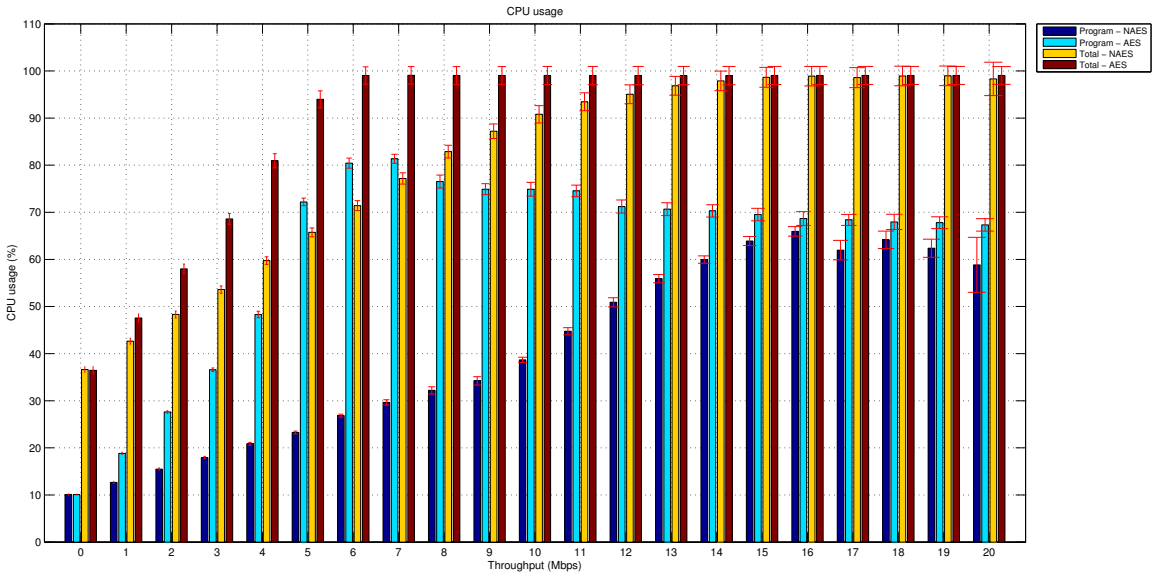



Figure 5.12: CPU usage measured in the Laboratory Testbed

program starts with a pool of memory reserved for queues, supplied by the `KLib` libraries. As the queues start flooding, the library starts resizing the queues occupying, therefore, more memory. If RED, or some congestion control mechanism, was not implemented, this memory would start increasing linearly with the traffic increase. RED allows to control this by limiting the number of packets present in memory simultaneously.

### 5.5.2 Real Testbed

The real network needed some way of monitoring the connections made by the nodes, because of that it was developed an interface with the server software that allowed to show the RSU where a node was connected, the number of hops it had to the server when last contact was made, and the time it would have passed since then, in milliseconds, as shown in Figure 5.15. In the figure, the upper right terminal has information on the state of the server, the lower right terminal was used to make throughput test, but since we wanted to know where handover occurred, that information was dumped in the upper terminal, and the lower left terminal, shows the interface made with the server, where nodes and information about them are shown. This information, was taken from the server using a UNIX Socket, and the columns represent EID, connected RSU IP, connected RSU Port, number of hops, and lastly, the time of last contact in milliseconds.

The first test made in Leixões was throughput. This throughput, as shown in Fig-

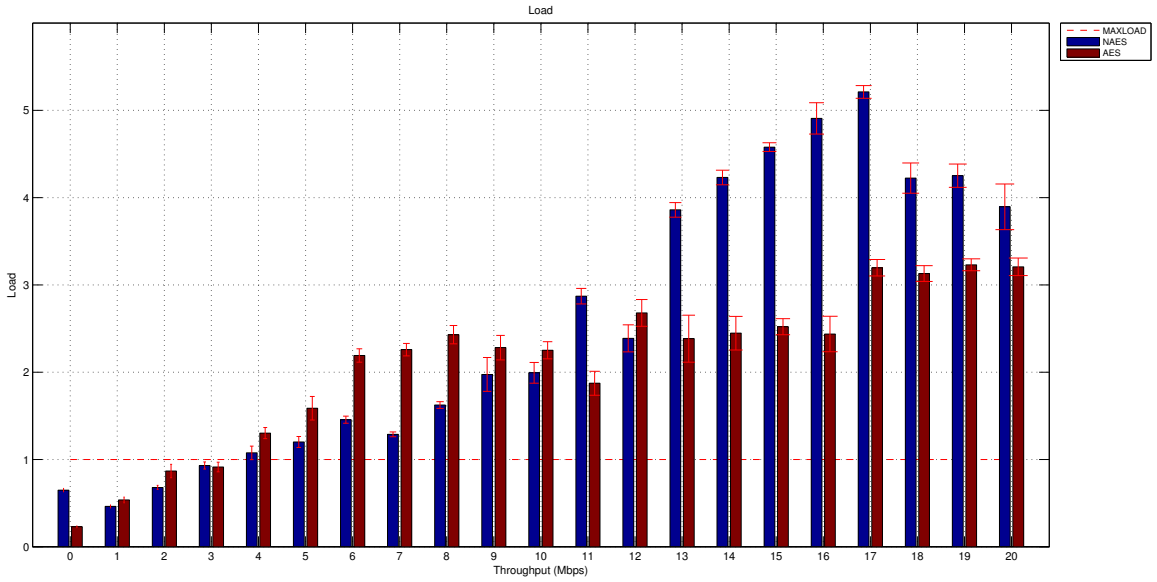



Figure 5.13: System Load measured in Laboratory Testbed

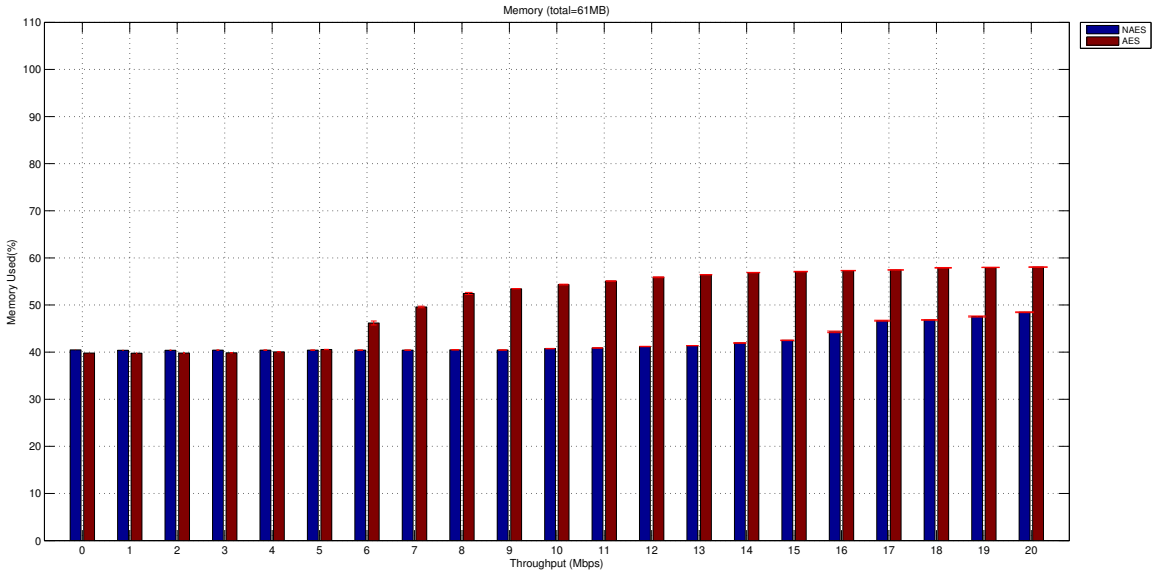



Figure 5.14: Memory usage measured in Laboratory Testbed

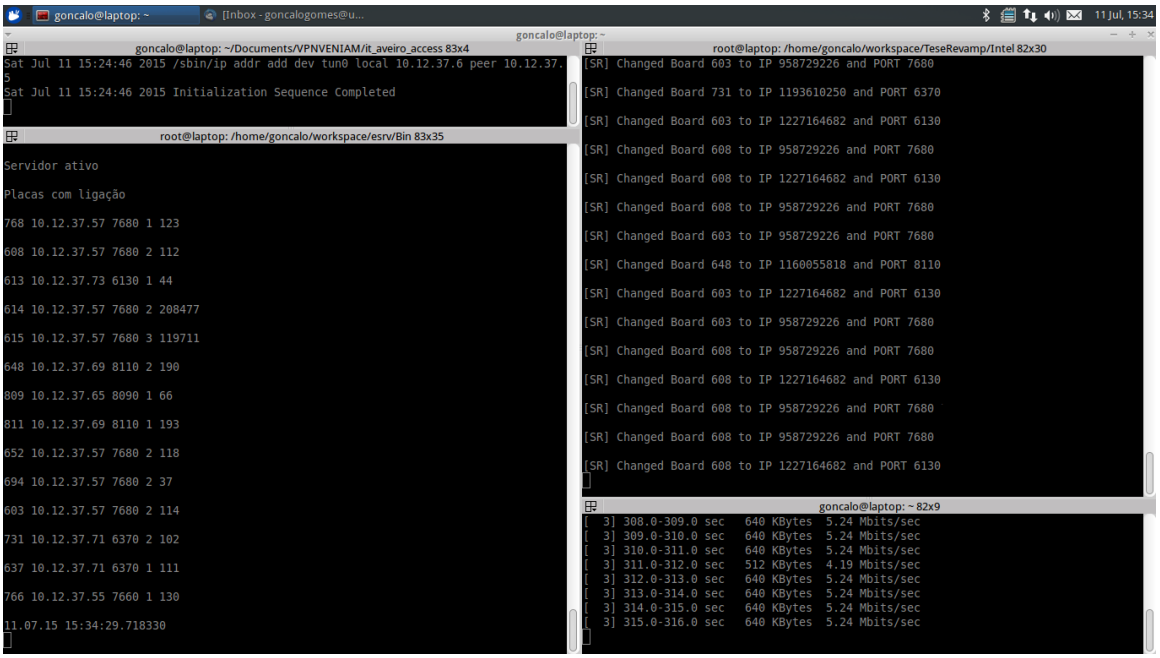



Figure 5.15: Screenshot of the setup used to get information in Porto de Leixões network

ure 5.16, in similarity to the one measured in the laboratory, is the maximum throughput in both modes (with and without the cipher).

Afterwards jitter was tested. The results are shown in Figure 5.17. The jitter on each hop is slightly greater than what was measured in the laboratory. The handover between infrastructure did not increase the jitter. However, on timeout it is seen that the jitter increases a lot on that instant. This is expected, because, for some time, the node has no information available to communicate to the server.

With respect to packet lack of order, we could see that it increases a lot as it was expected, as shown in Figure 5.18. In the laboratory, using the hysteresis mechanism it was not possible to get this information.

In the real network it was observed that the authentication time was similar to the one obtained in the laboratory, as shown in Figure 5.19 which was to be expected since that time depends mostly on the performance of the node and not the network by itself.

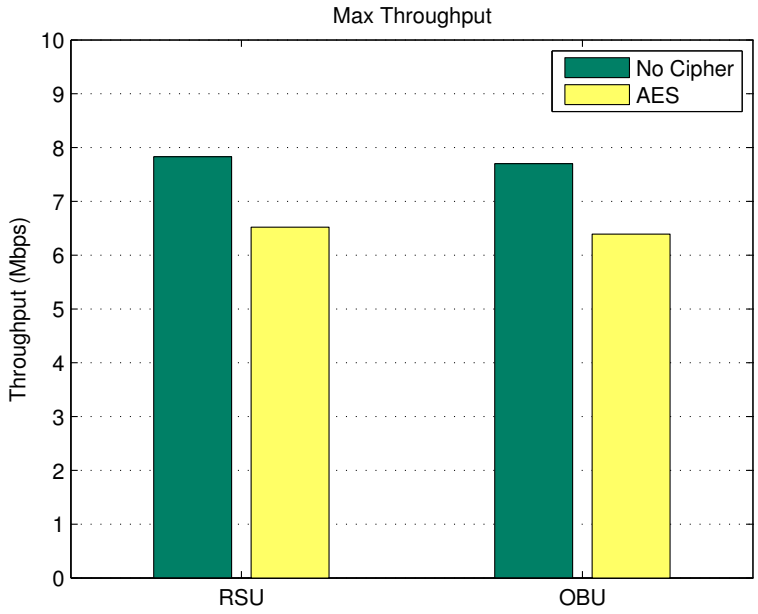



Figure 5.16: Maximum throughput obtained in Leixões

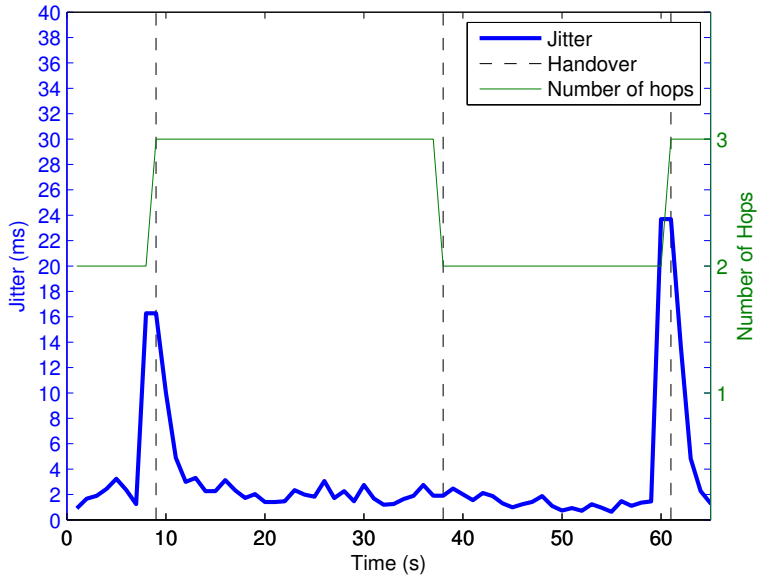



Figure 5.17: Jitter caused by the connection to the infrastructure and caused by the vehicular environment in Leixões

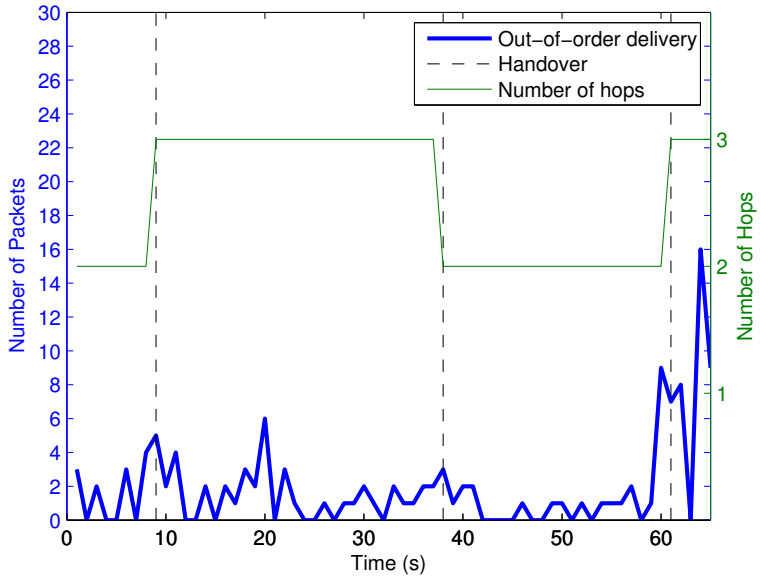



Figure 5.18: Packet disordering, caused by the duplicated packets and the route reconfiguration time in vehicular environment in Leixões

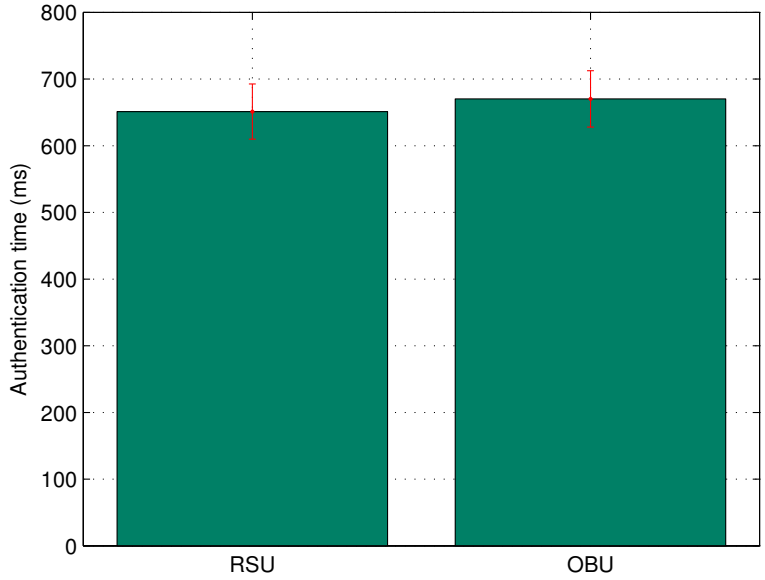



Figure 5.19: Authentication times in Leixões

# Chapter 6

# Conclusions and Future Work

This chapter reflects about the conclusions taken from the developed solution, and what can be done in the future to evolve this solution.

## 6.1 Conclusions

This dissertation proposed an architecture to provide secure connections between the vehicles in a vehicular network and an external network, in an efficient and scalable way, addressing seamlessly the vehicles' mobility. This solution proposed to remove protocol stacking overhead, and bind session parameters to a unique identifier, instead of the IP, which is normally utilized in the Internet solutions.

The security approach can authenticate both stakeholders of the network, which prevents MitM attacks. Also, the traffic is maintained confidential, and integrity is controlled during the session. Furthermore, the user sessions are maintained at all times, even when connectivity is not available. The routing scheme is also able to create a dynamic mesh network. By decentralizing routing, the network route reconfiguration made blackouts undetectable.

The security solution has been tested both in the lab and in a real vehicular network in the harbor of Leixões, Portugal. The obtained results show that the objectives have been met: (1) the authentication and the exchange of the key material is relatively fast, when compared to the lifetime of the keys, and RSA keys may be generated with bigger size to make factorization more difficult without turning the protocol infeasible; (2) furthermore, both endpoints, are able to maintain the keys and exchange data securely; finally, (3) the routing algorithm converges quickly and no blackouts were detected.

## 6.2 Future Work

The current solution can be improved in several ways, which would make the solution more resilient and versatile. An example of the improvements to this solution are listed as follows.

- Improve the routing scheme: The software works with a very simple routing strategy.
- Secure the routing scheme: The network is vulnerable to routing attacks. It should be assured that, at least, some form of source authentication will be implemented.
- Implement new cipher modes on the software in order to improve the throughput.
- Implement multi-homing and multipath.
- Reduce the number of packet copies in the software in order to improve performance.
- Implement the current solution in two modules embedded in the kernel space, one for the routing, and another for the security, minimizing the load imposed on the system by the excessive number of threads running, and using the kernel ring buffer to read and send packets.

The support for some of these improvements are already thought to some extent and implemented.

The cipher modes would be an easy extension, because the same approach which is used to negotiate if cipher is used or not, between the server and the nodes, could also be used to request a specific cipher algorithm.

Multi-homing, multi-path should be able to be implemented; however, in the current routing version, it seems too much decentralized to control the attachment points.

Vertical handover would also be interesting to test: the application should be able to work between 802.11p and 802.11g if there is an external software scanning and connecting to these networks. Cellular is, however, a different matter, since in the perspective of the OBU, the software should have to behave like an infrastructure connected using L3. At the moment, OBUs only use L2 communication.

The number of packet copies should be able to be reduced; however, in the current software version padding required at least one copy. There is one solution, however, which is only possible because we are in the control of the whole communication stack. The solution is to append the headers in the end of the user packet instead of the beginning.

This is counter-intuitive; however, if this approach is taken, the only operation on memory required will be the reallocation of the extra bytes.

# Bibliography


[1] IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band. *IEEE Std 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012 and IEEE Std 802.11aa-2012)*, pages 1–628, Dec 2012.

[2] Axsguard. Basic PPTP Concepts, July 2015.

[3] Dan Biebighauser. Testing random number generators, July 2015.

[4] Timo Bingmann. Speedtest and comparsion of open-source cryptography libraries and compiler flags, July 2008.

[5] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. Cryptology ePrint Archive, Report 2011/449, 2011. http://eprint.iacr.org/.

[6] A. Cardote. *Plataforma de Comunicações Veiculares com Infraestrutura de Suporte*. PhD thesis, Univ. de Aveiro, Aveiro, 2014.

[7] Broady Cash. Wave background information. Technical Report IEEE 802.11-04/0121r0, IEEE, January 2004.

[8] VPN Consortium. Vpn technologies: Definitions and requirements, July 2008.

[9] Magda El Zarki, Sharad Mehrotra, Gene Tsudik, and Nalini Venkatasubramanian. Security issues in a future vehicular network. In *European Wireless*, volume 2, 2002.

[10] P. Eronen. IKEv2 Mobility and Multihoming Protocol (MOBIKE). RFC 4555 (Proposed Standard), June 2006.

[11] ETSI. Intelligent transport systems (its);security;threat, vulnerability and risk analysis (tvra). Technical Report ETSI TR 102 893, ETSI, March 2010.

[12] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.

[13] Chunyao Fu, Zhifang Jiang, WEI Wei, and Ang WEI. An energy balanced algorithm of leach protocol in wsn. *IJCSI International Journal of Computer Science Issues*, 10(1):354–359, 2013.

[14] Lutz Gollan, Dr. Iur Lutz Gollan, Prof Dr, and Christoph Meinel. Digital signatures for automobiles?! In *in Systemics, Cybernetics and Informatics (SCI)*, 2002.

[15] Zygmunt J Haas and Marc R Pearlman. The performance of query control schemes for the zone routing protocol. *IEEE/ACM Transactions on Networking (TON)*, 9(4):427–438, 2001.

[16] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol (PPTP). RFC 2637 (Informational), July 1999.

[17] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic Routing Encapsulation over IPv4 networks. RFC 1702 (Informational), October 1994.

[18] R. Housley. Cryptographic Message Syntax (CMS). RFC 5652 (INTERNET STANDARD), September 2009.

[19] J.P. Hubaux, S. Capkun, and Jun Luo. The security and privacy of smart vehicles. *Security Privacy, IEEE*, 2(3):49–55, May 2004.

[20] Thomas Hühn. Myths about /dev/urandom, July 2015.

[21] AlgoSec Survey Insights. Examining the dangers of complexity in network security environments, October 2012.

[22] Intel. Intel® advanced encryption standard (AES) new instructions set, June 2014.

[23] J.T. Isaac, S. Zeadally, and J.S. Camara. Security attacks and solutions for vehicular ad hoc networks. *Communications, IET*, 4(7):894–903, April 2010.

[24] B. Kaliski. PKCS #7: Cryptographic Message Syntax Version 1.5. RFC 2315 (Informational), March 1998.

[25] 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. *Communications Surveys & Tutorials, IEEE*, 13(4):584–616, 2011.

[26] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (INTERNET STANDARD), October 2014. Updated by RFC 7427.

[27] S. Kent. IP Authentication Header. RFC 4302 (Proposed Standard), December 2005.

[28] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.

[29] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.

[30] T. Kivinen and M. Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526 (Proposed Standard), May 2003.

[31] B. Lloyd and W. Simpson. PPP Authentication Protocols. RFC 1334 (Proposed Standard), October 1992. Obsoleted by RFC 1994.

[32] Anahit Martirosyan, Azzedine Boukerche, and Richard W Nelem Pazzi. Energy-aware and quality of service-based routing in wireless sensor networks and vehicular ad hoc networks. *annals of telecommunications-annales des télécommunications*, 63(11-12):669–681, 2008.

[33] Andrew Mason. Ipsec overview part two: Modes and transforms, July 2015.

[34] Grant McWilliams. Hardware aes showdown - via padlock vs intel aes-ni vs amd hexacore, July 2011.

[35] Mohamed Nidhal Mejri, Jalel Ben-Othman, and Mohamed Hamdi. Survey on VANET security challenges and possible cryptographic solutions. *Vehicular Communications*, 1(2):53–66, 2014.

[36] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[37] D. Mitton and M. Beadles. Network Access Server Requirements Next Generation (NASREQNG) NAS Model. RFC 2881 (Informational), July 2000.

[38] Hassnaa Moustafa and Yan Zhang. *Vehicular networks: techniques, standards, and applications*. Auerbach publications, 2009.

[39] National Institute of Standards and Technology. Recommendation for block cipher modes of operation: Galois/counter mode (gcm), November 2007.

[40] H. Orman. The OAKLEY Key Determination Protocol. RFC 2412 (Informational), November 1998.

[41] Thomas Page. *The application of hash chains and hash structures to cryptography*. PhD thesis, University of London, 2009.

[42] Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Fisheye state routing: A routing scheme for ad hoc wireless networks. In *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, volume 1, pages 70–74. IEEE, 2000.

[43] Federal Information Processing Standards Publication. Announcing the advanced encryption standard (aes), November 2001.

[44] Ram Shringar Raw, Manish Kumar, and Nanhay Singh. Security challenges, issues and their solutions for vanet.

[45] Maxim Raya and Jean-Pierre Hubaux. The security of vehicular ad hoc networks. In *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 11–21. ACM, 2005.

[46] Sinchan Roychowdhury and Chiranjib Patra. Geographic adaptive fidelity and geographic energy aware routing in ad hoc routing. In *International Conference*, volume 1, pages 309–313, 2010.

[47] S. Sakane, K. Kamada, M. Thomas, and J. Vilhuber. Kerberized Internet Negotiation of Keys (KINK). RFC 4430 (Proposed Standard), March 2006.

[48] Bruce Schneier. A plea for simplicity, November 1999.

[49] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Performance comparison of the aes submissions, 1999.

[50] W. Simpson. The Point-to-Point Protocol (PPP). RFC 1661 (INTERNET STANDARD), July 1994. Updated by RFC 2153.

[51] W. Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994 (Draft Standard), August 1996. Updated by RFC 2484.

[52] OpenVPN Technologies. What is the difference between bridging and routing?, May 2014.

[53] OpenVPN Technologies. Openvpn, 2015.

[54] Northern Illinois University. Hashing, June 2015.

[55] Thibaut Vuillemin, François Goichon, Cédric Lauradoux, and Guillaume Salagnac. Entropy transfers in the linux random number generator. *hal-00738638, version*, pages 1–4, 2012.

[56] Yan Zhang, Jun Zheng, and Honglin Hu. *Security in wireless mesh networks*. CRC Press, 2008.

[57] G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759 (Informational), January 2000.

[58] G. Zorn and S. Cobb. Microsoft PPP CHAP Extensions. RFC 2433 (Informational), October 1998.

[59] A. Zúquete. *Segurança em Redes Informáticas, 4a Edição Actualizada e Aumentada*. FCA - Editora de Informática, Lda., 2013.

[60] A. Zúquete and C. Frade. Fast VPN mobility across Wi-Fi hotspots. In *Security and Communication Networks (IWSCN), 2010 2nd International Workshop on*, pages 1–7, May 2010.