



**UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH**

**Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona**

A PULL-BASED WIREGUARD CONTROL PLANE

A Master's Thesis

**Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de
Barcelona**

Universitat Politècnica de Catalunya

by

Alejandro Barcia González

**In partial fulfilment
of the requirements for the degree of
MASTER IN ADVANCED TELECOMMUNICATION
TECHNOLOGIES**

Advisor: Alberto Cabellos Aparicio

Co-Advisor: Jordi Paillissé Vilanova

Barcelona, May 2020



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH



Title of the thesis: A pull-based Wireguard control plane.

Author: Alejandro Barcia González

Advisor: Alberto Cabellos Aparicio

Co-Advisor: Jordi Paillissé Vilanova

Abstract

This thesis is based on IP security, Internet Protocol (IP) was developed without security or mobility. Later with the time, IPsec was implemented, a secure IP protocol which is complicated, weighty and it doesn't include mobility either. Also, new secure Virtual Private Networks (VPNs) appeared, but they had the same problem.

Wireguard is a VPN from the new security family protocols which provides a security layer for IP in a very simple way. Also, it includes mobility, a very useful features in the nowadays networks. But, Wireguard doesn't include a control-plane to distribute and manage all the cryptographic material into the devices.

We have used the Locator/Identifier Separation Protocol (LISP) to apply a Wireguard functional control-plane. In OOR software, where LISP is implemented, we merge all the Wireguard features with the LISP control-plane developing an efficient, functional and more important secure protocol. In the end, we achieve to include what Wireguard didn't have at the beginning, increasing the use cases for that new secure protocol.

Acknowledgements

I would like to acknowledge everyone who helps me during the thesis elaboration, in particular to my thesis director Albert Cabellos Aparicio who allowed me to make with him this investigation. Also, Jordi Paillissé Vilanova the co-director, who introduce me to Wireguard and LISP technology, also he helps me with all my doubts and questions. Finally, I would like to thank Albert López the developer of OOR software and who guide me through all the modification and development process during months. All of them have been essential for my thesis elaboration keeping in touch with continuous meetings, advances revisions, recommendations, etc.

To conclude, I would like to mention CISCO as the company who create the LISP protocol together with the UPC, they gave me the opportunity to work on this new advanced networking technologies. Also, the Computer Architecture department who gave me access to all the technology resources needed for my thesis simulation and experimentation.

Revision history and approval record

Revision	Date	Purpose
0	20/01/2020	Document creation
1	25/01/2020	Document revision
2	02/02/2020	Document revision
3	17/02/2020	Document revision
4	02/03/2020	Document finalization

Written by:		Reviewed and approved by:	
Date	20/01/2020	Date	31/03/2020
Name	Alejandro Barcia González	Name	Albert Cabellos Aparicio
Position	Project Author	Position	Project Supervisor

Table of contents

Abstract	1
Acknowledgements	2
Revision history and approval record.....	3
Table of contents	4
List of Figures	6
List of Tables	8
1. Introduction.....	9
1.1. Objectives	9
1.2. Methods and Procedures.....	10
1.3. Work Plan.....	10
1.3.1. Study the LISP and Wireguard Protocols.....	11
1.3.2. Discuss a LISP+Wireguard Implementation.....	11
1.3.3. Study the LISP Code	12
1.3.4. Wireguard integration in LISP	12
1.3.5. Data Collection	13
1.4. Gantt Diagram	13
1.5. Deviations and Eventualities.....	14
2. State of art of the technology	16
2.1. Locator/Identifier Separation Protocol (LISP).....	16
2.2. Open Overlay Router (OOR)	18

2.3.	Wireguard.....	19
3.	Methodology / project development	23
3.1.	Design a secure Wireguard control-plane using the LISP technology.....	23
3.1.1.	Secure LISP Control Plane communications with Wireguard.....	23
3.1.2.	Secure LISP+WG control-plane.....	25
3.2.	Designed secure protocol development.....	29
4.	Results	31
4.1.	Throughput.....	31
4.2.	Delay to Add an xTR map cache entry	32
4.3.	Map-Server response	34
4.4.	End-to-end Delay.....	34
4.5.	Handover.....	36
5.	Budget.....	38
6.	Conclusions and future development.....	40
	Bibliography.....	41
	Appendices.....	43
	Glossary	50

List of Figures

Figure 1.1: Secure LISP data communications channel.....	9
Figure 1.2: Wireguard Secure Control-Plane.	10
Figure 1.3: Work Plan diagram.	10
Figure 1.4: September Work Plan.....	13
Figure 1.5: October Work Plan.	13
Figure 1.6: November Work Plan.....	14
Figure 1.7: December Work Plan.....	14
Figure 1.8: January Work Plan.	14
Figure 1.9: February Work Plan.....	14
Figure 2.1: LISP Communication Overview [14].....	16
Figure 2.2: OOR project evolution [19].....	18
Figure 2.3: SDN Overlay using OOR [19].	19
Figure 2.4: Wireguard Logo [17].	20
Figure 2.5: Wireguard Cryptokey Routing Table (CTR) [13].....	20
Figure 2.6: Protocol Overview [13].....	21
Figure 2.7: Packet constructed with Wireguard protocol.	21
Figure 2.8: wg interface configuration.	22
Figure 2.9: wg peer configuration.....	22
Figure 3.1: Security structure for xTRs and Map-Server.	23
Figure 3.2: Control-Plane key public key distribution.....	24

Figure 3.3: Control devices interfaces for secure control-plane.....	24
Figure 3.4: Secure LISP control-plane diagram.	25
Figure 3.5: Security structure for the LISP data-plane.....	26
Figure 3.6: Security features for the three proposals.....	28
Figure 3.7: Proxy Reply key distribution system design.	28
Figure 3.8: Security Key LCAF defined in RFC 8060 [4].	29
Figure 3.9: Final LISP security model using Wireguard.....	30
Figure 4.1: Throughput graphic.....	32
Figure 4.2: Add MN map cache entry graphic.....	33
Figure 4.3: Add MN map cache entry CDF graphic.....	33
Figure 4.4: Map Server response to Map-Request with different cache sizes.	34
Figure 4.5: End-to-end delay graphic.....	35
Figure 4.6: End-to-end delay CDF graphic.....	35
Figure 4.7: Handover time graphic.....	37

List of Tables

Table 1.1: WP1, study LISP and Wireguard Protocols	11
Table 1.2: WP2, LISP + Wireguard Implementation.....	11
Table 1.3: WP3, study LISP code	12
Table 1.4: WP4, Wireguard integration with LISP	12
Table 1.5: WP5, Data collection.....	13
Table 2.1: Use of each Wireguard Header.....	20
Table 5.1: Human Resources budget for Engineer work.....	38
Table 5.2: Human Resources budget for the thesis directors.....	39
Table 5.3: Budget for Software and Hardware.....	39
Table 5.4: Final budget considering all the proposed costs.....	39

1. Introduction

The presented thesis is based on Internet Security, since the moment that the Internet came to light, security has been one of the main concerns in the community. When IP was released, the Internet communication took a big step forward but, it didn't include security. Many investigators tried to design a security protocol which gives IP the security needed. Then, IPsec was developed, it was a complex protocol which uses certificates, authorized key managers, certificates authorities, etc. It is until now a complex and heavy protocol, so during the years, new security protocols appeared trying to improve IPsec.

One of these new security protocols is Wireguard [13], it is a simple, fast and modern VPN protocol created to improve existing protocols like IPsec or OpenVPN. Wireguard includes all the IPsec functionality in an easy and user-friendly way. Moreover, IPsec can't deal with IP mobility which is a common situation in nowadays networks, but Wireguard is designed to deal with these situations, hence, it manages IP mobility scenarios by itself without the user interaction. Finally, it is a cross-platform protocol which can be used in Windows, MacOS, BSD, iOS, Android and Linux. However, one of the main problems of Wireguard is that it doesn't have a control-plane or a platform to distribute all the cryptographic material needed to perform the secure communication channels. Since today, these materials must be distributed manually by the user and this is the focus of the present thesis.

To design a Wireguard control-plane, we will use the Locator/Identifier Separation Protocol (LISP) [1]. It is a fully operative and functional protocol created by CISCO with the UPC. Which makes LISP relevant is that it offers a standard inter-domain and dynamic overlay, it follows a map-and-encap approach where overlay identifiers are mapped to underlay locators. So, the overlay traffic is encapsulated in the locator space and then it is routed through the underlying network. But, what is relevant for us, is that LISP uses a control-plane to distribute the identifier-locator mapping and we will use it to perform a Wireguard control-plane too.

As a consequence, if we use LISP to design a Wireguard control-plane, we will implement a LISP security protocol too. By default, LISP doesn't include security in their communications, we can find related works where they try to include some kind of security protocol like LISP-SEC [10] or LISP-CRYPTO [8]. However, these protocols are not good enough or they are so complicated. Therefore, using Wireguard we can perform an efficient and useful LISP security protocol.

1.1. Objectives

The final objective for my thesis is to create a secure Wireguard control-plane using the LISP technology, Figure 1.2. So, in order to do it, we need to understand how both protocols work by themselves, then we will combine them to perform a useful and efficient communication protocol (LISP+Wireguard). Thus, we will have all the benefits and improvements of LISP in the OOR networks plus the security that Wireguard applies to IP communications, Figure 1.1.



Figure 1.1: Secure LISP data communications channel.

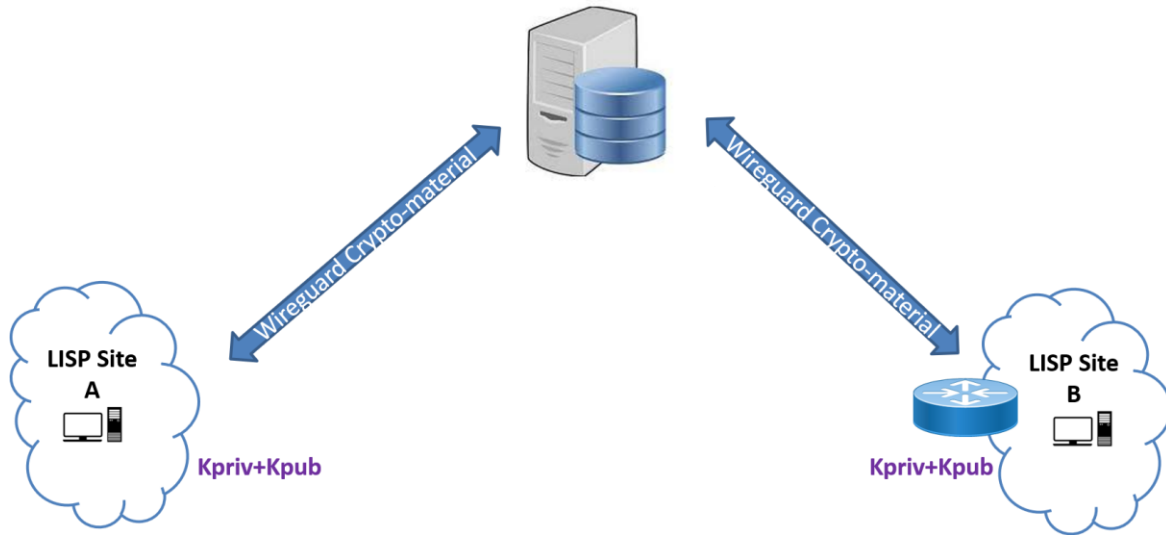


Figure 1.2: Wireguard Secure Control-Plane.

1.2. Methods and Procedures

To combine both protocols LISP and Wireguard, we need to use the LISP original code. As we have mentioned before, LISP is a developed and nowadays used protocol so, we will need to modify this code to include all the Wireguard functionalities. Thus, once we get familiarized with the code and understand how it works, we will perform all the necessary changes until we can obtain one reliable working protocol. The Open Overlay Router (OOR) [14] project is where LISP is implemented in C and it's was created in the UPC (Universitat Politècnica de Catalunya) by Albert López with CISCO.

Finally, when we get a functional protocol, we will obtain different measures of Throughput and Latency to compare them with the original protocol which doesn't include security. Then, we will answer questions as; Do we improve the existing LISP protocol? How much we lose to gain security? it is reliable and useful? Etc.

1.3. Work Plan

We have followed a work plan model during the thesis elaboration to classify the different parts of my project and follow an organized work structure.

Does are the work packages:

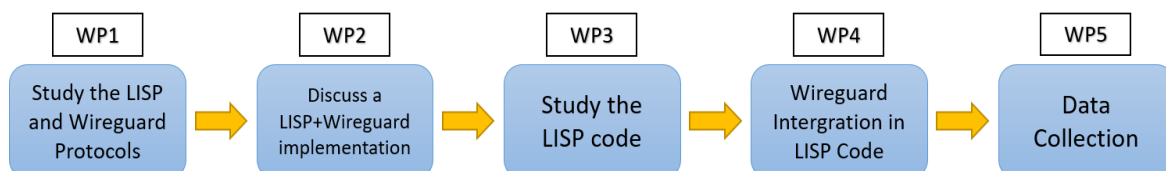


Figure 1.3: Work Plan diagram.

In every work package, there are different specifications to follow and internal activities, we can define and summary each of them in the following tables.

1.3.1. Study the LISP and Wireguard Protocols

Project: Study the LISP and Wireguard Protocols	WP ref: (WP1)
Main Component: Reading / Investigation	
Brief description: Obtain all the information about both protocols, learn how they work and how they are used.	Planned Start Date: 06/09/2019 Planned End Date: 06/10/2019
	Start: 06/10/2019 End: 02/10/2019
Internal Work T1.1: Read LISP RFCs. Internal Work T1.2: Read the Wireguard Paper.	

Table 1.1: WP1, study LISP and Wireguard Protocols

1.3.2. Discuss a LISP+Wireguard Implementation

Project: LISP+Wireguard Implementation	WP ref: (WP2)
Main Component: Investigation	
Brief description: Propose and design different methods to include Wireguard in LISP protocol, discuss all of them to check the benefits and the issues of each implementation.	Planned Start Date: 07/10/2019 Planned End Date: 25/10/2019
	Start: 03/10/2019 End: 08/11/2019
Internal Work T2.1: Think about different ways to include Wireguard inside LISP. Internal Work T2.2: Evaluate the pros/cons of the proposed methods. Internal Work T2.3: Select the best method that would be implemented.	

Table 1.2: WP2, LISP + Wireguard Implementation.

1.3.3. Study the LISP Code

Project: Study the LISP Code	WP ref: (WP3)
Main Component: Programming	
Brief description: Get familiarized with the Source LISP code and understand how it is composed, which functions are useful, where are implemented all the configurations, etc.	Planned Start Date: 28/10/2019 Planned End Date: 25/11/2019
	Start: 11/11/2019 End: 29/11/2019
Internal Work T2.1: Read the LISP code. Internal Work T2.2: Understand how it is composed. Internal Work T2.3: Check which are the useful functions and files that I will modify to include the new code.	

Table 1.3: WP3, study LISP code

1.3.4. Wireguard integration in LISP

Project: Wireguard integration in LISP	WP ref: (WP4)
Main Component: Programming	
Brief description: Modify all the necessary LISP code to include the Wireguard functionalities and protect the protocol communications.	Planned Start Date: 26/11/2019 Planned End Date: 27/12/2019
	Start: 02/12/2019 End: 20/01/2020
Internal Work T2.1: Modify LISP Code. Internal Work T2.2: Apply Wireguard functionalities in the Code.	

Table 1.4: WP4, Wireguard integration with LISP

1.3.5. Data Collection

Project: Data Collection	WP ref: (WP5)
Main Component: Simulations and Tests	
Brief description: Obtain different measures of Throughput and Latency doing some test and simulations with the new working protocol. Discuss the results and analysed it to get a final conclusion.	Planned Start Date: 07/01/2020 Planned End Date: 31/01/2020
	Start: 21/01/2020 End: 12/02/2020
Internal Work T2.1: Perform different simulations. Internal Work T2.2: Obtain all the useful data and create graphics to analyse it. Internal Work T2.3: Discuss the results.	

Table 1.5: WP5, Data collection

1.4. Gantt Diagram

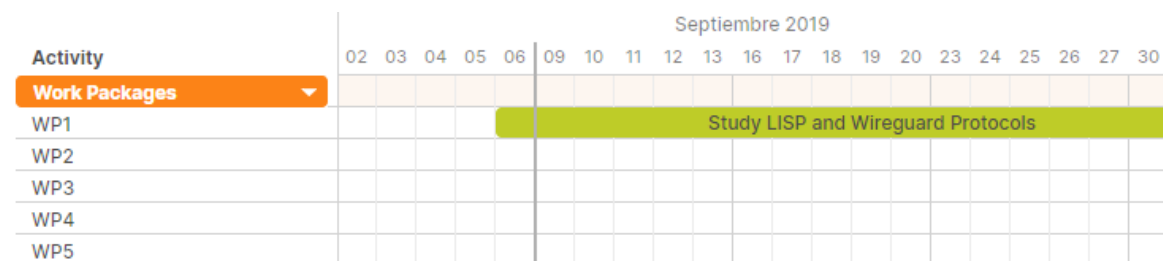


Figure 1.4: September Work Plan.

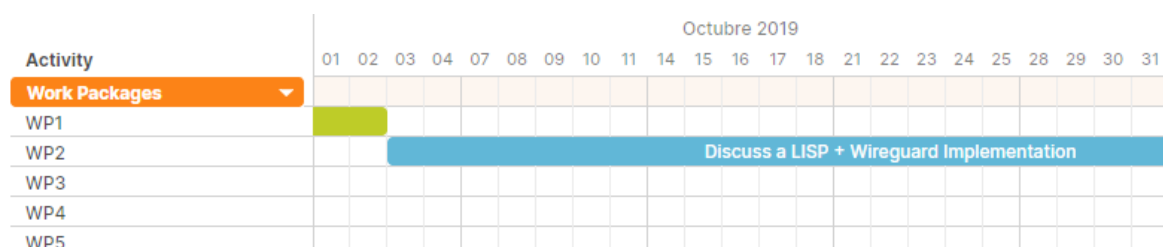


Figure 1.5: October Work Plan.

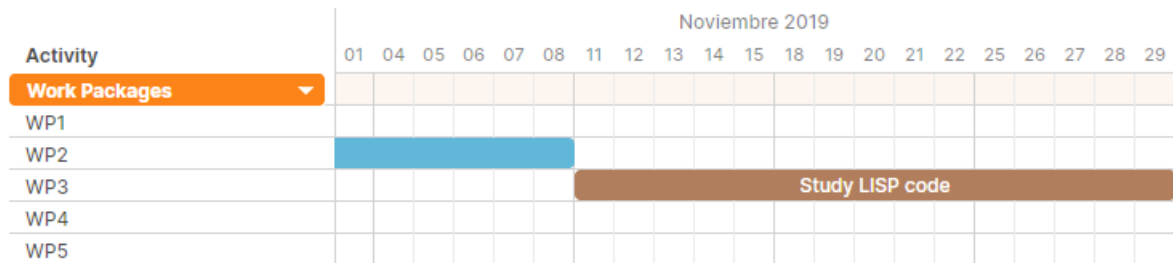


Figure 1.6: November Work Plan.

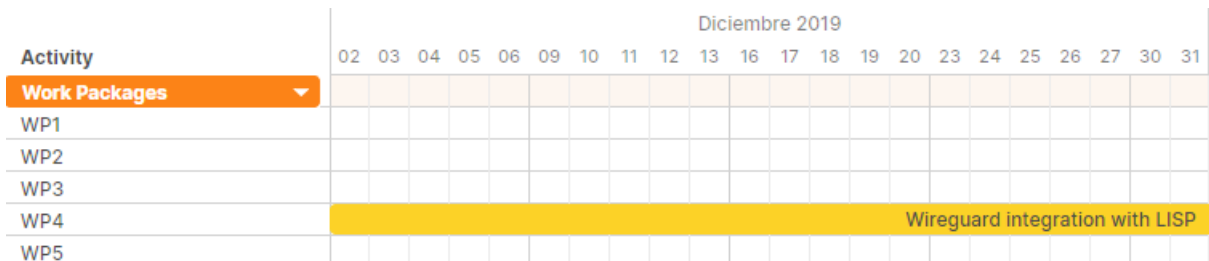


Figure 1.7: December Work Plan.

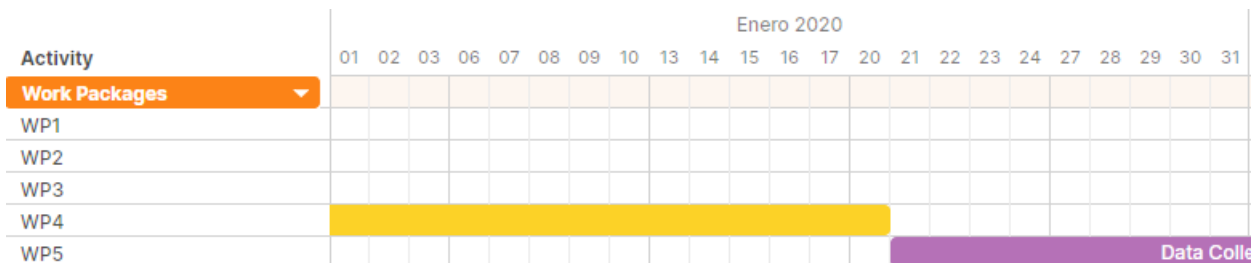


Figure 1.8: January Work Plan.

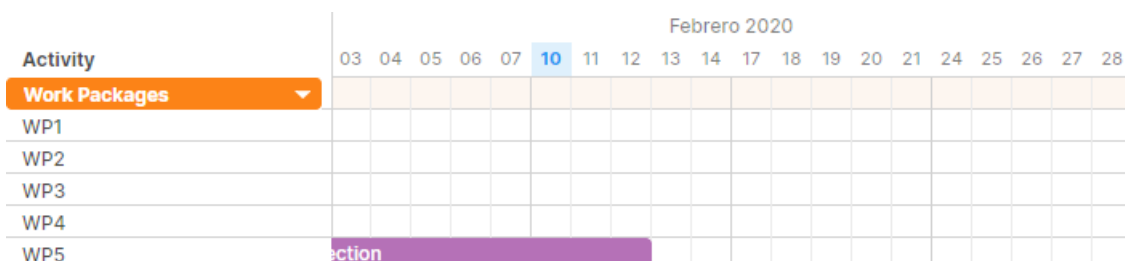


Figure 1.9: February Work Plan.

1.5. Deviations and Eventualities

We found different eventualities during the work packages which deviates me from the initial planned dates. Therefore, we are going to explain them classified by work package where they take place.

- WP2-Discuss a LISP + Wireguard implementation.

When we had to design possible implementations to merge both protocols, we face some difficulties because we had to fit as possible the LISP methods and characteristics. This means that not any security protocol or implementation was valid to apply in the worked scenario. Some of the purposes were not good enough or they just are not functional in terms of cost and hardware. So, there were fail proposals which didn't fit into the LISP protocol.

- WP3-Study LISP code.

In this package, we took contact with the LISP software implementation in OOR. As LISP is a huge and complex protocol, the code was not easy to understand. When we were studying it, I needed a lot of help from the software developer Albert López. He knew how was the code structured and which was the use of every function that I didn't understand.

- WP4-Wireguard integration in LISP code.

The integration of Wireguard in LISP took me so long, this is because to be sure that the integration was successful and the security protocol worked well, we did so many tests with virtual machines. We found lots of corner cases that we solved during the integration and when we solved one another appeared. This was a slow process which takes me most of the thesis time.

- WP5-Data Collection

In the final package, we needed to obtain data from the working protocol to make some graphs. Here, each test needed a particular scenario to be configured, we had problems configuring one scenario in particular, the Handover. This scenario required to configure different Wi-Fi interfaces in one machine and also configure two access points (AP). When all the scenario was configured, we realised that there were problems with the firewall because we were working in different networks, and those networks were not open between them in the firewall. That's why I talked with Albert López who was in charge of manage the networks that we have been using during all the tests and he solves it.

2. State of art of the technology

There is a huge background behind in my thesis, this is because we are dealing with one developed and standardized protocol LISP, a functional Programmable Overlay Network software OOR which implements the protocol and also, a secure VPN protocol like Wireguard. It's necessary to review the literature of all in order to understand what is the purpose of my work.

2.1. Locator/Identifier Separation Protocol (LISP)

The Locator/Identifier Separation Protocol (LISP) [12] is a protocol developed by CISCO Systems with the UPC and standardized by the Internet Engineering Task Force (IETF) in 2013 RFC6830. It is a routing architecture which provides a new semantic for IP addressing. In current networks architectures, the IP routing and addressing uses a single numbering space, the well know IP address, to define two pieces of information, device identity and the way the device attaches to the network. LISP defines the overlay via separating the host identity from its location, it creates two different namespaces: Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). Hence, every host is identified by one EID and its network point of attachment by an RLOC. The two namespaces are flexible, they can be the typical IPv4 or IPv6 but also, MAC, GPS location, etc.

The packet routing is based on EIDs located in LISP sites, and on RLOCs in the transit networks. So, in LISP sites where all the EIDs are located, there are some edge nodes called xTR, their function allows the transit between EIDs and RLOC spaces. To do it, the protocol uses a map-and-encap method, EIDs are mapped to RLOCs and the xTRs encapsulate the EID packets into the RLOC traffic. To know which is the destination RLOC of a packet, LISP provides a public Mapping System where all the couples EID-to-RLOC are stored and available to the xTR. This Mapping System is composed of Map-Resolvers (MR) and Map-Servers (MS), the Map-Server is where the peers are stored and the Map-Resolvers finds in which Map-Server the peer is stored.

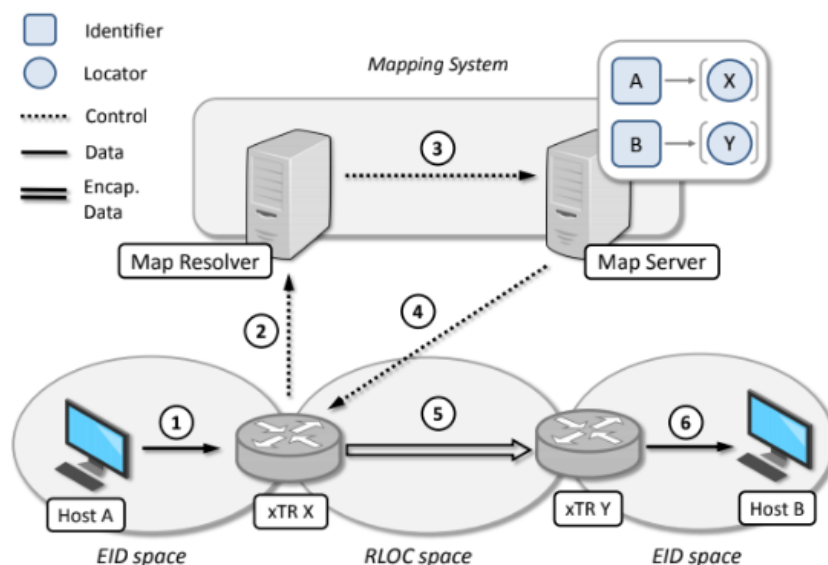


Figure 2.1: LISP Communication Overview [14]

The LISP topology is divided into two planes, the Data-Plane where the communication is established and includes the EIDs and RLOCs devices. On the other hand, the Control-Plane which is the Mapping-System, including here the Map-Servers and Map-Resolvers. The typical communication diagram is represented in Figure 2.1, one host A defined by one EID A wants to communicate with another host B defined by other EID B and located in a different LISP site. The host A will send the packets to his edge xTR X router, the xTR X will ask for the peer EID B to the Map-Resolver in the Control-Plane, this control message is called Map-Request. The Map-Resolver will find in which Map-Server is located the searched EID and it will forward the obtained Map-Request from the xTR X. Once the Map-Request reaches the Map-Server, it will answer to the xTR X with another kind of control message called Map-Reply, this message has the mapping information about EID pref-to-RLOC. At this moment, the xTR X is able to encapsulate the data packet to xTR Y which decapsulate and forward it to the host B. This working method is called Proxy Mode, where the Map-Server is who responds to the Map-Request with the Map-Reply. But there is another mode where the response is made by the xTR Y, in this case, the Map-Server will also forward the Map-Request to the xTR Y where the asked EID B is located, then the same xTR Y will generate the Map-Reply response to the xTR X. It will save the mapping in his cache and when a new packet with destination EID B arrives, it will be routed through the mapped RLOC.

There are other relevant devices as Proxy xTR used to connect with non-LISP sites, Re-Encapsulating Tunnel Routers (RTR) to enforce in-path policies and also, LISP Mobile Nodes (MN) where the EIDs and RLOCs are included in the same device so connections are preserved across handover events. Even, exist the possibility for the Map-Server to work as a Map-Resolver too. So, the messages will go directly to the Map-Server/Resolver [2].

LISP was originally designed to face one of the most important problems on the Internet nowadays, the continued growth of the Border Gateway Protocol (BGP) routing tables. With the LISP namespace subdivision, EIDs are allocated in sites without the provider interaction, this means that they are independent but more important, they are not advertised on the global internet. In the end only the RLOCs are those which are advertised and included in the BGP routing tables, then providers would have the possibility of highly aggregate them, and help scale these routing tables. Moreover, dynamic mapping of EID-to-RLOC enables programmable overlays, also, it helps in the migration of IPv4 to IPv6 equipment's in Cloud Systems or big Data Centres because it allows to use different types of namespaces, for example, use IPv4 to the EIDs space and IPv6 for the RLOCs space.

In terms of security, as we mentioned before, there are different drafts from the IETF drafts which try to include some kind of security into the LISP protocol. LISP-CRYPTO (draft-ietf-lisp-crypto) [8] includes confidentiality into the LISP Data-Plane, but it doesn't make a reference about the Control-Plane, it was first published in February 2014 and became an RFC in February 2017, RFC8061. Time after the publication of that RFC, another one called LISP-SEC (draft-ietf-lisp-sec) [10] came out, the last update was in January 2020. This one tries to solve some unresolved problems from the previous work, for example, the security of the Control-Plane. This new approach to perform a feasible and functional security protocol was not good enough, it has some missing details that makes it not completely secure like the communication between the Map-Resolver and

the Map-Server which is not taken into account. Nowadays, there are people in CISCO and UPC dealing with the security implementation of LISP, thus, there is still work to do in this research area.

2.2. Open Overlay Router (OOR)

The Open Overlay Router [19] was released in December 2015 as a rename of another project called LISPmob.org. This project was initially developed as a LISP mobile node implementation but soon became a fully featured LISP implementation. This has a good impact in the community so, researchers, startups and companies started using it in different ways, as a result, it was added support for other data and control planes. With this, the project evolved and became an overlay router supporting protocols well beyond LISP, changing the name by the Open Overlay Router (OOR) in 2016. It was developed by Albert López Brescó of the Universitat Politècnica de Catalunya (UPC) and we can find the updated project in his GitHub page [19] where is explained all the necessary to install and work with it.

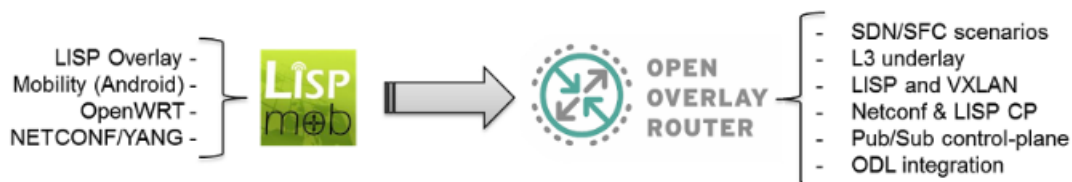


Figure 2.2: OOR project evolution [19].

OOR supports two encapsulations for data-plane as VXLAN-GPE and LISP, as well as control-planes: NETCONF with YANG and LISP control-plane. The project is in continuous development, so it aims to provide more data and control planes in the future.

As we said, OOR allows users to easily deploy programmable overlay networks. This means that we can find so many different and interesting use cases, for example, SDN. The easiness of deployment and the capacity to be remotely programmed make OOR a valuable choice to serve as the data-plane for SDN networks. It allows bypassing the constraints of the physical underlying network to program policies directly in the instantiated overlay. Therefore, it enables flexible over-the-top solutions that reduce deployment complexity, costs and time. Moreover, due to its multi-platform support, it can be used to offer a homogeneous control of the network despite the heterogeneous physical network beneath. Note that OOR is agnostic to the underlying networking equipment, which means that SDN networks enabled by OOR can be deployed both on top of SDN-aware hardware or over non-SDN legacy appliances. OOR support for home routers and mobile nodes, enables OOR to effectively achieve SDN deployments at the edge.

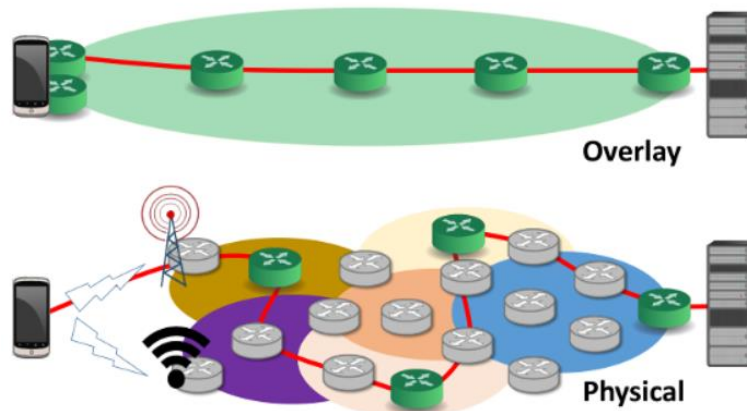


Figure 2.3: SDN Overlay using OOR [19].

It also can be used for service function chaining (SFC) so, it can process and forward traffic with different encapsulations. The OOR nodes can be deployed at the edge of a service function path or in the same path. Another interesting service is dynamic VPN provisioning, it can connect remote sites over transit networks using different encapsulations technologies and dynamically managing these connections remotely. In terms of security, we can leave OOR to handle it or use a third-party solution. There are other features and use cases like easy home multihoming or IPv6 transition among others.

As we can see, OOR is a tested and functional software to provide programmable overlay networks, although it is quite new and continuously evolving trying to incorporate new and outstanding features.

2.3. Wireguard

Wireguard is a modern VPN extremely simple and fast which uses state-of-the-art cryptography [13]. The developers say that it aims to be faster, simpler, leaner and more useful than IPsec, also, it is created to be considerably more performant than OpenVPN. Initially, it was released for the Linux kernel, but now it is a cross-platform VPN (Windows, Android, iOS, MacOS, BSD) and widely deployable. It is still in development, but already it seems to be the most secure, easiest to use and simple VPN solution in the current industry. Recently, it has been added to the mainline Linux kernel increasing the routing power and the functionality of the protocol.

This VPN is easy to configure and deploy as SSH. A VPN connection is made exchanging the public keys of the endpoints like in SSH but, all the rest is transparent for the user because it is handled by Wireguard. Even, it is made to deal with IP roaming so, we can change our endpoint IP and Wireguard will update it keeping active the communication. This means that we don't need to manage connections, be concerned about the actual state, manage daemons, or worry about what is happening under the hood. Moreover, Wireguard presents an extremely basic and user-friendly interface, we can configure one end-to-end encrypted tunnel in less than a minute, that's why it is becoming so popular and powerful between users and investigators.

Additionally, Wireguard uses more advanced and powerful cryptographic algorithms like Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF. Also, it is developed in very few code lines, consequently, it is easier to audit for security vulnerabilities compared with other protocols like IPSec or OpenVPN/OpenSSL. Another security feature is that it is designed to be a silent protocol which makes it even more secure in front of different attacks.



Figure 2.4: Wireguard Logo [17].

Finally, we can stand out the high performance achieved by this VPN, the combination of extremely high-speed cryptographic primitives and the fact that Wireguard is in the Linux kernel means that secure networking can be very high-speed and make it suitable for small embedded devices like smartphones and fully loaded backbone routers.

So, as a brief resume, Wireguard is interesting for our project because it is simple, fast and provide an easy and fast mobility method. Moreover, it provides a total disjoint between his headers making it more flexible and compatible with another kind of protocols.

Header	Use
Outer Header	Mobility
Wireguard Header	Security
Inner Header	Firewall

Table 2.1: Use of each Wireguard Header.

Wireguard is a VPN protocol which uses keys (public and private) to provide security as many other protocols that we have talked before like SSH or OpenVPN, and it does this under UDP. But, in this case, it uses a Cryptokey Routing Table (CRT). Hence, Wireguard uses the IPs plus the keys to configure his routing table, Figure 2.5.

Configuration 1b

Interface Public Key	Interface Private Key	Listening UDP Port
HIgo...8ykw	yAnz...fBmk	41414
Peer Public Key	Allowed Source IPs	Internet Endpoint
xTIB...p8Dg	10.192.122.3/32, 10.192.124.0/24	
TrMv...WXx0	10.192.122.4/32, 192.168.0.0/16	
gN65...z6EA	10.10.10.230/32	192.95.5.64:21841

Figure 2.5: Wireguard Cryptokey Routing Table (CTR) [13].

In order to route packets, the protocol checks the table before sending a packet and after receiving it. So, if one host wants to send a message to another one, it must have the IP as usual, but now, a public key of this destination host too. If the IP or the public key are not in the table, the packet will be dropped. Additionally, we can define an Internet endpoint, if the packet needs to be sent through the other networks.

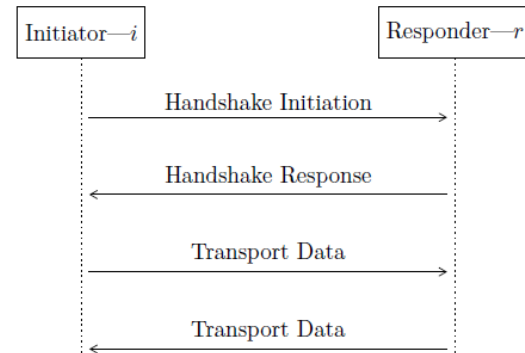


Figure 2.6: Protocol Overview [13].

The protocol only has 3 types of messages as we can see in Figure 2.6, first of all, handshake messages to share the cryptographic information to derive the secure keys used after in the data packets. Then, the data packets where the information is encrypted and completely secure from the attackers.

The Wireguard packet is constructed by an outer header which contains the source and destination endpoint, the UDP header which is the used transport protocol, then the Wireguard header where we find the security information and finally the encrypted data packet.



Figure 2.7: Packet constructed with Wireguard protocol.

The user can configure easily one secure tunnel between Wireguard peer following these steps: (We will use a Linux example) [15]

1. Download the Wireguard package from the repository which is included already in Linux. **`sudo apt install Wireguard`**
2. Now, we can use the Linux net command to set up a new Wireguard interface. **`ip link add dev <itf_name> type Wireguard`**
3. Then, we should define an IP related with the newly created interface using the Linux commands too. **`ip address add dev <itf_name> <ip>/<mask>`**
4. Once the interface is created, we should define a private key for it using the Wireguard commands. **`wg set <itf_name> private-key ./<file_private_key>`**
 - We can obtain the keys using the function that Wireguard gives to the user or with other valid methods.
 - (private key) **`wg genkey > <file_private_name>`**
 - (public key) **`wg pubkey <file_private_name> > <file_public_name>`**

5. The user can check the created interface using the **wg** command Figure 2.8.

```
root@kali:~# wg
interface: wg0
public key: vawi0C0tBk0GsdKZ/Chn8F/jtMrTLjtNdhfSTJv3GAQ=
private key: (hidden)
```

Figure 2.8: wg interface configuration.

6. At this moment, we can configure the Wireguard peer using the Wireguard commands. For example, to configure one peer to the interface, we will need the peer public key, IP, port and endpoint. **wg set <itf_name> peer <peer_public_key> allowed-ips <wg_peer_interface_ip> endpoint <peer_endpoint>:<peer_endpoint_port>**

```
root@kali:~# wg
interface: wg0
public key: vawi0C0tBk0GsdKZ/Chn8F/jtMrTLjtNdhfSTJv3GAQ=
private key: (hidden)

peer: WmB0zWYZDdGRinjTWNljaGbcixwbWb1BQ8zCHdMLzg=
endpoint: 192.168.1.150:2345
allowed ips: 10.0.1.0/24
```

Figure 2.9: wg peer configuration.

7. Finally, in the other peer, the user should configure the same and both peers could have secure communications between them.

With these few steps any user would be able to create a secure tunnel communication between two hosts and for that reason, Wireguard is so powerful by its simplicity and user-friendly implementation.

3. Methodology / project development

We are going to explain how the project was developed, in which parts are divided and what kind of methodology we have followed in each one.

3.1. Design a secure Wireguard control-plane using the LISP technology.

The first part of the project was designing a feasible and functional security protocol for Wireguard control plane, using the features that LISP gives us. Starting with the control-plane because to establish a LISP data tunnel we need first to exchange some messages through the control-plane. Once the control-plane was designed, we moved to the data-plane.

3.1.1. Secure LISP Control Plane communications with Wireguard.

In order to secure the LISP control-plane, we should provide to the xTRs and the MS with a key pair, public and private for each one of them as we can see in Figure 3.1. This is mandatory because as we have said before, Wireguard creates secure tunnels between peers using a technology based on keys. The purpose is obtaining a secure control-plane for all the control messages, Map-Register, Map-Request, Map-Notify and Map-Reply.

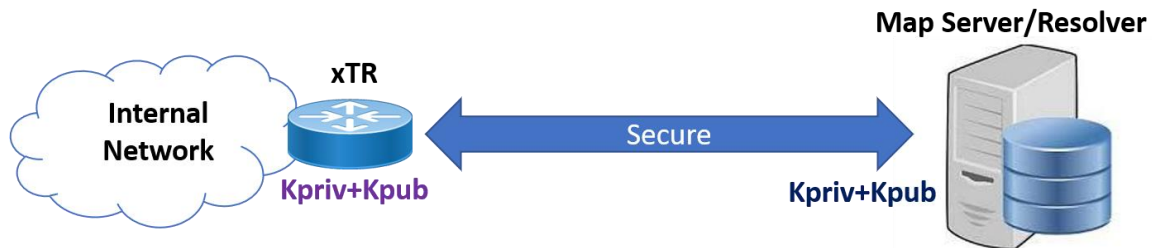


Figure 3.1: Security structure for xTRs and Map-Server.

In this scenario, we decided to distribute the keys manually, it can seem unfunctional or unusual but, to establish a secure tunnel with Wireguard both peers should have the public key of the other, without the public key the communication won't work. The reason to do it manually is that we didn't want to leave the assignment of sharing the keys to a third party or use a certification authority as happens in the current internet. The process would be more complicated and we would need a trusted certification authority to provide valid keys/certificates to the xTRs. So, in my design, every time that one xTR is added to the network, it should have the public key of the MS in the configuration file and the MS the public key of the xTR too.

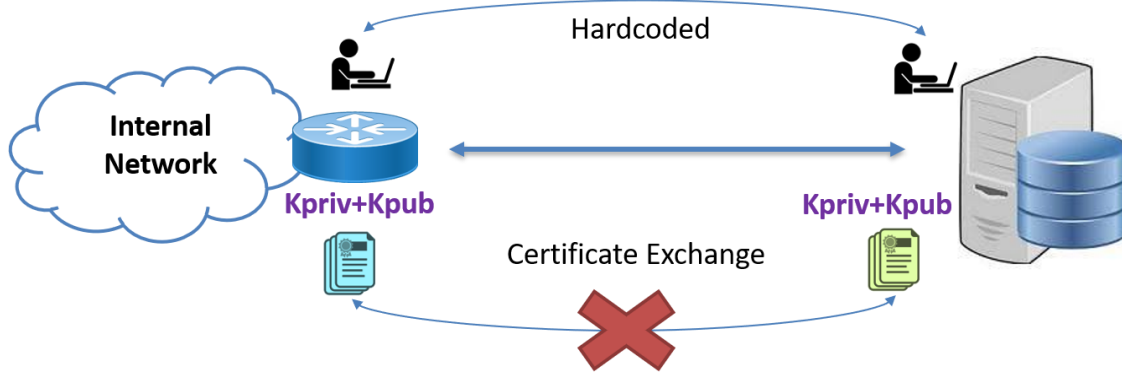


Figure 3.2: Control-Plane key public key distribution.

Once we decided to distribute public key in control-plane manually, we faced another problem. Wireguard works with its own interface rather than the typical eth0 or default interface as we have seen before, hence, we need to configure control-ips for all the devices, we call control-ip to a new local user-configured ip for the Wireguard interface. In the end, we will have an xTR with one interface for Wireguard, another one for LISP and the default one which acts as the endpoint. However, the Server will only have two, the Wireguard and the endpoint interface.

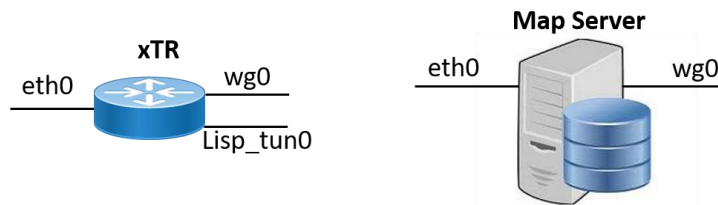


Figure 3.3: Control devices interfaces for secure control-plane.

With all these modifications, we will obtain a secure control-plane communication, all the control packets from the xTRs to the MS will be routed through the Wireguard interface, where they will be encrypted and later, they will go out to the network through the default interface. Thus, when the packet arrives at the endpoint MS interface, it will be passed to the Wireguard one where it will be decrypted.

The Wireguard routing table would be equal to the current one. The xTR will have the public key associated with the MS mapped with his control-ip, additionally, it would have the server endpoint which is the IP that should be routed to the network in the outer header, Table 3.1.

The MS table will have every xTR RLOC public key mapped with his control-ip Table 3.2. In this case, it wouldn't be necessary to configure an endpoint because the communication is always established by the xTR. As in every communication Client-Server, the clients must know to which endpoint IP is the server connected but, the server doesn't need it. Therefore, when the communication is initiated by the xTR, the MS will use the obtained source IP from the outer header as the most recent endpoint used by the xTR, thus, all the response packets will be routed to this endpoint. If the xTR changes its endpoint for whatever reason, the MS will update the routing table with the new one.

Interface Public Key	Interface Private Key	Listening UDP port
Higo....9kyk	Yanz....7z7k	41414
Peer Public Key	Allowed Source IP	Internet Endpoint
Qs1A....xulT	Server Ctr-IP	Server Endpoint

Table 3.1: xTR Wireguard routing table.

Interface Public Key	Interface Private Key	Listening UDP port
Qs1A....xulT	D4rT....hG67	41414
Peer Public Key	Allowed Source IP	Internet Endpoint
Higo....9kyk	RLOC Ctr-IP	
Vg5h....mV10	RLOC Ctr-IP	

Table 3.2: Map-Server Wireguard routing table.

With this proposal, we will obtain a completely secure LISP control-plane communications using Wireguard, solving one of the main problems of the LISP security implementations studied until today. LISP-SEC tries to establish secure communication with the control-plane, but as we have said, it was not well designed making it unfunctional. So, in our scenario, the control-plane is strongly secure in front of any kind of third-party or possible attacker, Figure 3.4.

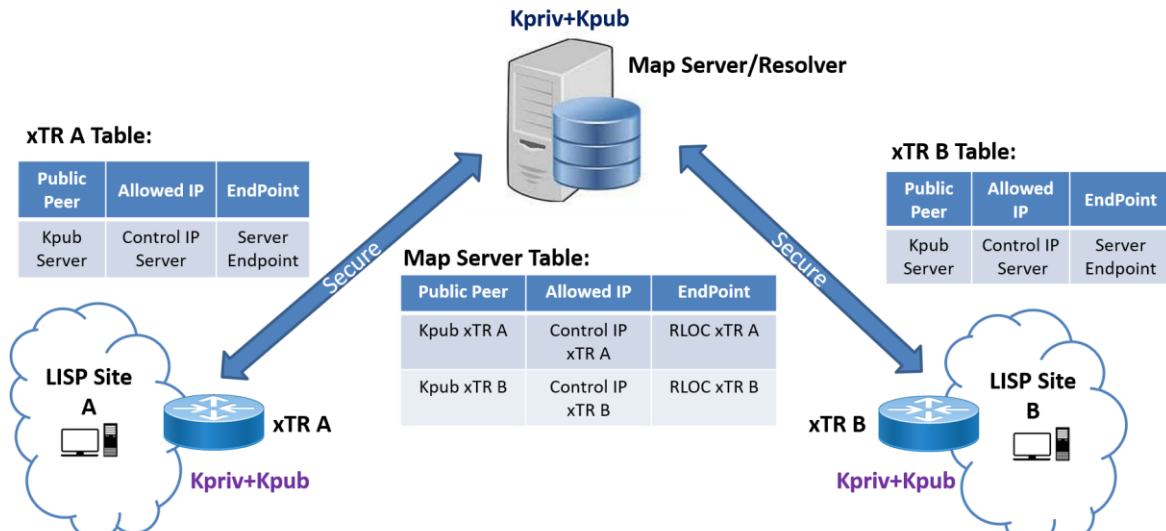


Figure 3.4: Secure LISP control-plane diagram.

3.1.2. Secure LISP+WG control-plane.

At this moment, with a design of secure LISP control-plane communications, we start to think about how to implement a secure Wireguard control-plane with LISP. We know already that this protocol works with keys, hence, we need to provide a new key pair for the data-plane which will be associated to an EID now.

Here, we have to differentiate into the two LISP working methods.

- xTR as an edge LISP site device where the EIDs and RLOCs spaces are connected. The traffic is generated by the EIDs and the xTR act as a router.

- xTR as a Mobile Node (MN) where the EIDs and RLOCs are located in the same device. The traffic is generated by the xTR itself.

But in terms of security, it doesn't matter in which kind of mode we are working on, because the device security configuration will be the same.



Figure 3.5: Security structure for the LISP data-plane.

In contrast with the control-plane, the Wireguard data-plane key distribution system shouldn't be done manually, it would be cumbersome because we would have to manually add a key for each new xTR, it would be unfunctional and inefficient. So, in this case, we designed a key distribution system focused on the MS, taking advantage of the already secure control-plane.

LISP uses the MS as a mapping database for the peers of EIDs-to-RLOCs, so we thought to use this model including the security features. Hence, the MS will map now the relation of **EID&Kpub-to-RLOC**. With this update, when any xTR ask for the RLOC associated with one EID, it will ask for the public key associated with the destination EID too. Thereby, the MS will become the data-plane key distribution manager and of course, we will use the control-plane to provide this system.

In order to design different models, we take into account the LISP packet exchange, applying all the Wireguard security features and implement a key distribution system modifying as less as possible the LISP protocol.

So, let's take a brief reminder about how control-plane works. Every time that one xTR is added to the network it sends a **Map-Register** to update the MS Cache with the mapping of his EIDs-to-RLOCs, when the MS updates its tables, it sends back a **Map-Notify** to the xTR. This is the first part which the control-plane is used, now it has to differentiate two working methods.

- The normal LISP control-plane packet exchange:

When we want to establish communication between A and B, the xTR A sends a **Map-Request** asking for the EID B to the MS. It will check two things, first, if the EID requested is registered and second, if it has the mapping of the requested EID-to-RLOC. If it has the mapping in his cache, then it forwards the Map-Request message to the xTR B with the mapping in a **Map-Request-Forward**. And finally, the xTR B sends a **Map-Reply** with the mapping to the xTR A, closing the process.

- The Proxy Mode LISP control-plane packet exchange:

This work method is nearly the same explained just before, but with the difference that here who responds to the Map-Request is directly the MS. So, in the same scenario, we want to establish communication between A and B. The xTR A will ask for one EID sending a **Map-Request** to the MS. As before, MS will check if the requested EID is registered in his database and check his cache looking for the mapping EID-to-RLOC. If the mapping is in his cache, it will answer with a **Map-Reply** to the xTR A closing the process.

Knowing how the LISP control-plane messages works, we designed three key distribution approaches trying to modify minimally the protocol. First of all, there is one requirement that all the designed methods have to accomplish, when an xTR sends a Map-Register, in this message should be included the EID public key. That means, now every time that we register an EID-to-RLOC into the MS, the xTR will include the EID public key to be also saved into the MS cache. Now, we will explain the three different designed methods.

- 1. New Map-Server message:** In this approach, when an xTR A sends a Map-Request asking for an EID, the MS will answer with a non-existent “Map-Reply” which includes the requested EID public key. Then, when the MS forwards the request of the xTR A, the message should include the source EID public key. What’s left is not modified and follows the typical LISP process where the MS forward the request of the xTR A to the xTR B and itself will send the Map-Reply. As we can see, with this method we are including a new message from the MS to the source xTR and also, adding the source EID public key in the Map-Request-Forward message.

Both keys will be secure because they go always through the control plane which is already secured by Wireguard but, the mapping EID-to-RLOC which goes into the Map-Reply directly from xTR B to xTR A will be unsecure, what is an issue if we want to build a completely secure protocol. We can see the designed diagram in appendices 1.1.

- 2. Proxy Reply:** With this method, we are using the Proxy mode. So, here we take benefit of the directly response by the MS to the xTRs Map-Requests. When xTR A sends a Map-Request to the MS, it will answer with all requested data through a secure channel. The Map-Reply for xTR A will include the EID-to-RLOC mapping and the destination EID public key too. Moreover, in this case, a new message from the MS to the destination xTR B called “Map-Notify” will be added, this message will include the source EID public key which is needed to establish the secure tunnel communication.

Using this approach all the control-communication will be encrypted and secure through the control-plane, the mapping and the keys will be completely secure. Figure 3.6 shows how would be the implementation of this approach.

- 3. Map-Server Signatures:** This is another approach where we thought to use signatures to provide Map-Reply authenticity. Hence, the key distribution process would be as follows.

First, the source xTR A sends a Map-Request to the MS asking for the destination EID. Then, the MS will build and forward a message to xTR including the source EID public the destination EID public key signed by his private key. Signing the public key means that anybody with access to the server public key can see the signed key, but no one can change/modify it because the signature would change. When the xTR B receives the forwarded message, it will answer to the source xTR A with a Map-Reply message including the destination EID public key in clear and the one signed by the MS.

Finally, the source xTR A would get an authentic public key, and it can check the key validity making sure if the signed key and the original are the same. We can check how the signature/digest and validity process works in appendices 2.

This method is not using the proxy reply working mode and brings off a secure protocol to establish a secure data tunnel, but we would not provide confidentiality to the xTR public key. There is a diagram model in appendices 1.2 for more information.

Proposal Nº 1		Proposal Nº 2		Proposal Nº 3	
Security	Yes/No	Security	Yes/No	Security	Yes/No
Integrity	✓	Integrity	✓	Integrity	✓
Authenticity	✓	Authenticity	✓	Authenticity	✓
Confidentiality	✓	Confidentiality	✓	Confidentiality	✗

Figure 3.6: Security features for the three proposals.

From all the designed key distribution methods, the one that we consider to be the best and we have implemented is the **Proxy Reply** method. The main advantage is that we don't send packets through the data plane, only the control plane, (recall that is secured by the Wireguard control plane tunnel). Also, the MS is always in charge of distributing the keys, not xTRs. The disadvantage is that we work with the limitations of the Proxy mode, for example, we lose the node reachability, if xTR B is down xTR A would not know it until after a while. Analysing all the advantages and disadvantages, we conclude that it is the most efficient and simple way to implement the protocol.

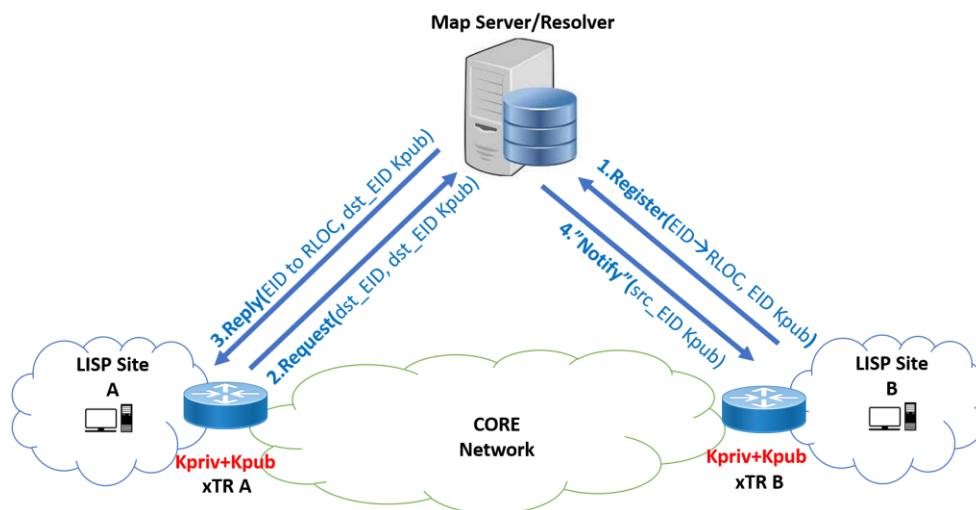


Figure 3.7: Proxy Reply key distribution system design.

3.2. Designed secure protocol development.

Once we decided which method to implement, we started working into the software development of the design. As we mentioned, the OOR project implements the LISP protocol and it is entirely written in C. So, with the help of Albert López, who is the developer of the OOR project, I started modifying parts of the code.

To start, we created a new LISP Canonical Address Format (LCAF) which defines how is to transport a security, Figure 3.8. The security LCAF was already defined in the RFC 8060 [4] where are defined all the other LCAFs.

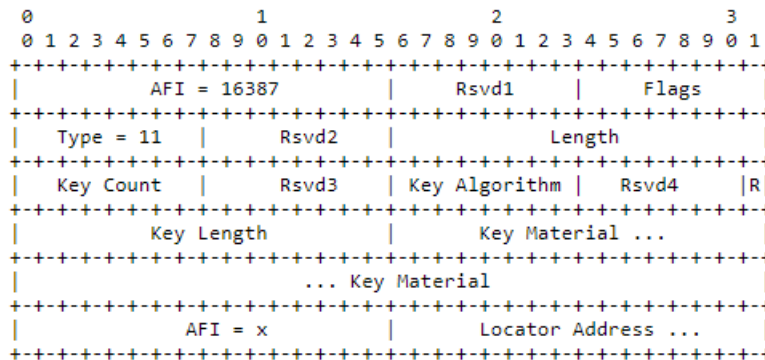


Figure 3.8: Security Key LCAF defined in RFC 8060 [4].

This LCAF was not implemented in the OOR code, so if we were going to work with security keys this had to be programmed. Hence, when the LCAF was ready, we developed some functions to associate one IP with one key.

Then, we began to work in the configuration file. When OOR is executed, it uses a configuration file where that defines some necessary data about to configure the xTR, MN, MS or RTR. To add the Wireguard security features in the OOR software, we needed to modify this configuration file including some data like device control-address, private key, public key, etc. We modified two LISP devices: the xTR and the MS. Appendix 3. contains all this changes. At that time, with all the new data in the configuration file, we changed the code to read all these data. With all the necessary data in my hands, we were able to start developing the control-plane and data-plane Wireguard tunnels.

First, we started with the control-plane, as we have said in previous points, the control-plane should be secure before any message was sent. For this reason, the xTR had all the information about the MS public key and endpoint IP address in the configuration file, and the MS had the public key of all the xTRs in the network too. So, we developed new code that configures the Wireguard control plane tunnel just after OOR start-up. This way, any new control-packet like Map-Register or Map-Request will be sent securely through the Wireguard interface. Also, we modified the Map-Register because now, it saves in the MS cache the peer EID-to-RLOC plus the public key associated with that EID.

After, we developed the secure LISP data-plane. This one was more complicated than the control-plane because, it included the key distribution design that we had selected. The process begins when an xTR sends a Map-Request to the MS. The MS answers this request with a Map-Reply including the asked EID-to-RLOC and also the public key associated to the requested EID. With all this data, the xTR which sent the Map-Request

has all the necessary information to create his Wireguard data-plane secure tunnel. We programmed the necessary code creating this tunnel when the xTR received the Map-Reply. At this moment, one of the two xTR was completely configured but we need to configure the other too. Therefore, we created a new message generated by the MS when it received one Map-Request. Using the existing Map-Notify which by default is used to answer the Map-Register, but in this case, it will be used to notify the destination xTR that other xTR has asked for him. The message includes the EID who asked and the RLOC and the public key associated with this EID. Thus, as before, the destination xTR has now all the data to create his secure Wireguard data-tunnel. Once the two xTR are completely configured, the communication between them flows through the data-plane Wireguard interface closing the implementation.

To conclude, we did some tests to verify the functionality. We tested the code using the Mobile Node (MN) and the xTR configuration, and both worked as expected. Also, we checked if the packets were really encrypted and protected through WireShark, a traffic analyser, more information in appendix 4. Hence, we got what we had been looking for from the beginning, a new security model for the LISP control and data plane as we can see in Figure 3.9.

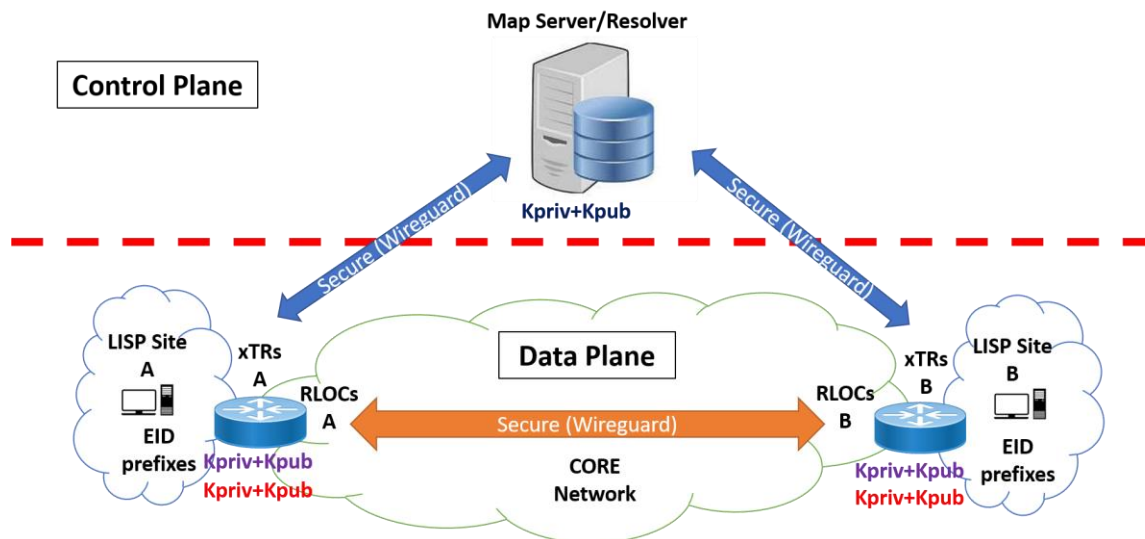


Figure 3.9: Final LISP security model using Wireguard.

4. Results

To check how efficient was our OOR LISP security model, we had to test different values and compare them with the OOR model without security. The features that we decided to test are Throughput, time adding an xTR map cache entry, response time of the MS to Map-Register with different cache sizes, end-to-end delay and handover time.

To replicate the real scenario in the best conditions, we used three machines with the same characteristics to act as two xTR and one MS. The machines Operative System (OS) are Linux 4.4.0 with 16G of RAM and 24 CPU working in 32 or 64 bits. All of them were connected over a dedicated Gigabit Ethernet network. we did all the tests in this scenario unless the handover.

We manipulated the machines remotely using SSH. OOR was compiled for Linux and installed in the three machines and the results were obtained using different BASH scripts. Each script was used for different intentions so, the data collection method was different too.

4.1. Throughput

In order to measure the maximum throughput supported by both models OOR and OOR+Wireguard, we used the *nuttcp* tool, it generates UDP packets with a user-selected size and we monitored the input and output rates. Using two MN and one MS, we ran *nuttcp* as a Server in the recipient MN, in the other one (the sender), we ran the *nuttcp* command to send UDP packet.

- Packet originator: *nuttcp -u -i 1 -R10M -l 1362 -T 60 -v 11.0.0.2*
- Server: *nuttcp -S*

We used two different packet sizes for both models because the headers are not the same using Wireguard or the simple OOR model, we are encapsulating a LISP packet into Wireguard and it makes increase the header. So, if we want to get the maximum throughput with no fragmentation, the packet data length had to be one specific size according to the MTU which was 1500 bytes. Therefore, we used for the original OOR model packets of 1388 bytes, and for our secure OOR model 1380 bytes.

We can observe the results in Figure 4.1, showing how our model using OOR+Wireguard gets a higher throughput than the original one. In our 1 Gigabit Ethernet network the maximum throughput that we can get would be 1Gbps but, as we can see both models arrive to a saturation point where they can't get more. For the secure OOR model the saturation point is in 900Mbps and the original one is between 700-800 Mbps.

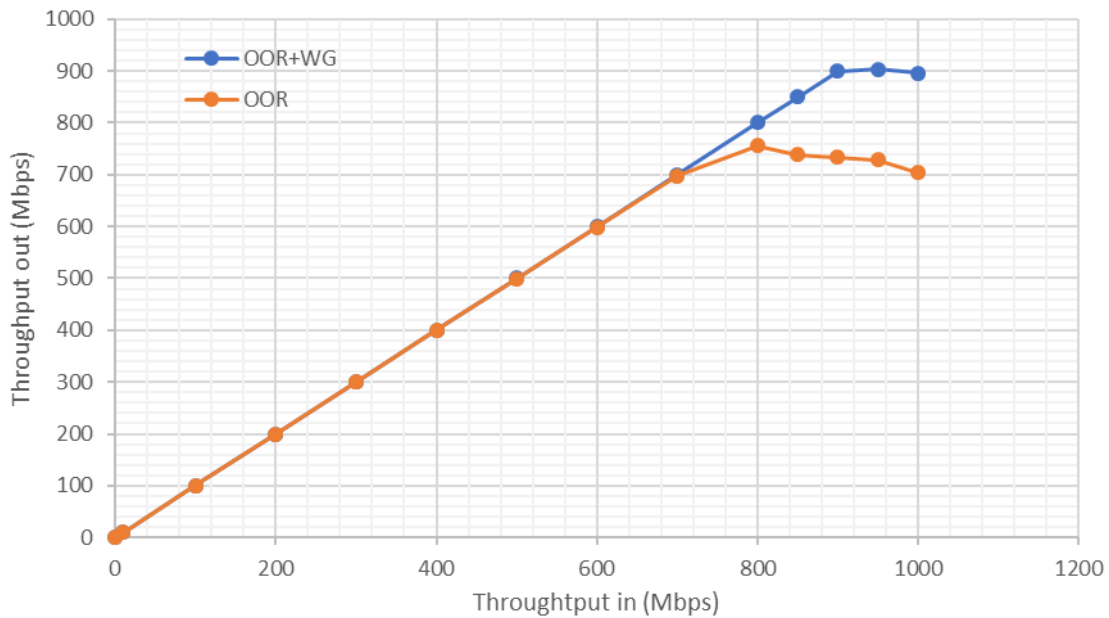


Figure 4.1: Throughput graphic.

The secure OOR model using Wireguard offers more throughput than OOR for more than 150 Mbps. This improvement is due to the Wireguard Linux Kernel implementation so, the routing is faster than the original LISP protocol. In the end, we are comparing the routing thorough the user-plane which is faster by default with the control-plane. So, making use of this secure VPN, we can increase the maximum throughput in our network, making it more efficient.

4.2. Delay to Add an xTR map cache entry

In this test, we analysed the time to add an entry in the xTR cache when it asks for a destination EID. The scenario was two MN and one MS, then one MN tries to send a message to the other one sending a Map-Request, the MS answers with a Map-Reply and the entry of this EID is added into the MN cache.

We measured the time since the moment that the MN didn't find the entry of the EID in his cache until the moment it was added. We repeated this test 100 times for our secure OOR model and the simple one.

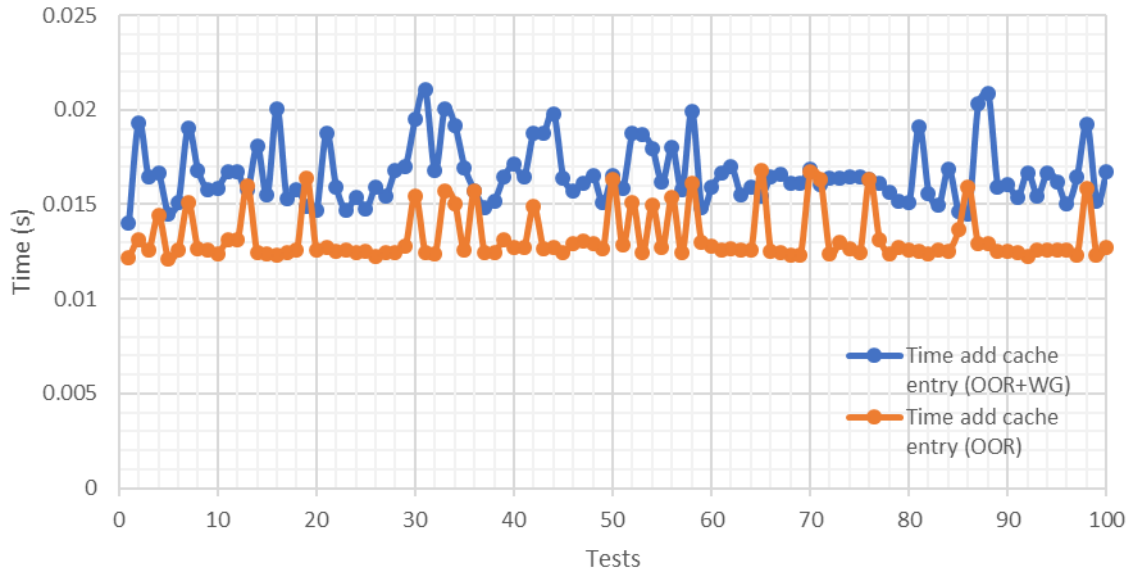


Figure 4.2: Add MN map cache entry graphic.

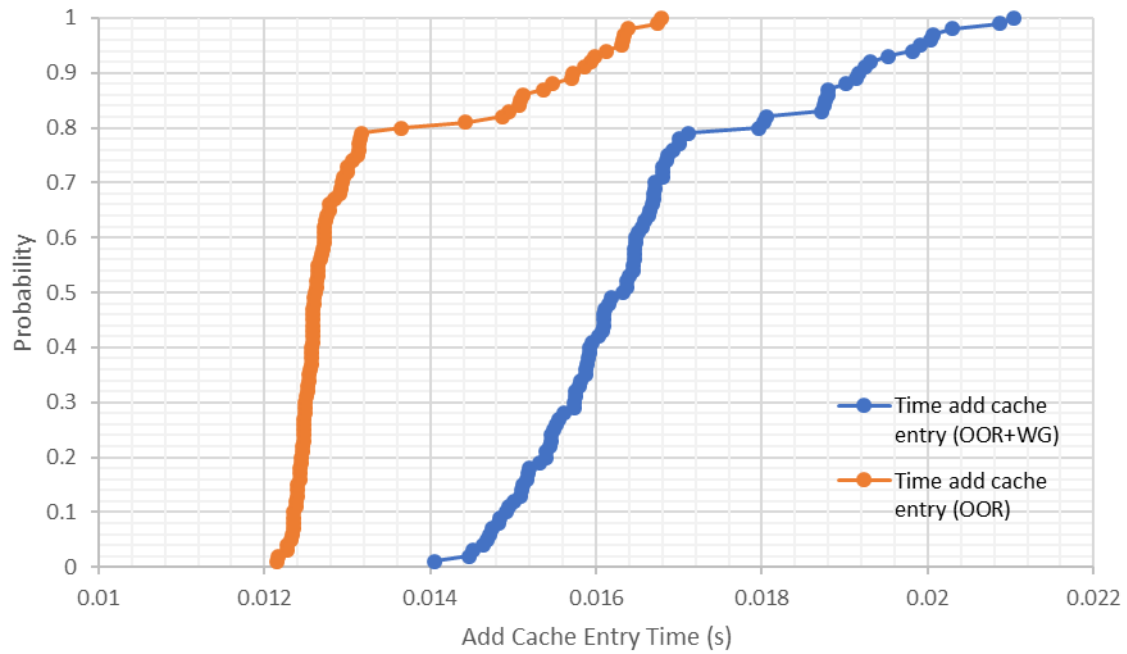


Figure 4.3: Add MN map cache entry CDF graphic.

As Figure 4.2 and 4.3 show, the time to add one entry in the MN cache is higher for our secure model than the other one. We expected this result because adding one entry in the MN cache, it also creates the Wireguard interfaces in the data-plane for the new EID. This means that we always spend more time adding entries in our model. However, we can see that the difference is not so big, the average value for the OOR model is 13.23931 milliseconds and for the OOR+WG model 16.61133 milliseconds. This result show what we had expected since the beginning, with the secure OOR model we lose some small time adding entries in the xTR caches, in this case 3.37202 milliseconds.

4.3. Map-Server response

The following experiment is used to measure the answer speed of the MS when it is asked for an EID. Here, we measured the time since the MS received one Map-Request until it responded with a Map-Reply, we repeated this test 50 times with different cache sizes from 10 to 100000 for both models.

To implement the Map-Request 50 times we used *lig*, a program which allows us to produce artificial Map-Requests. By default, it waits until the Map-Request is answered with a Map-Reply to send a new Map-Request but, we changed this working method to produce as many Map-Request as I wanted without having to wait for the response.

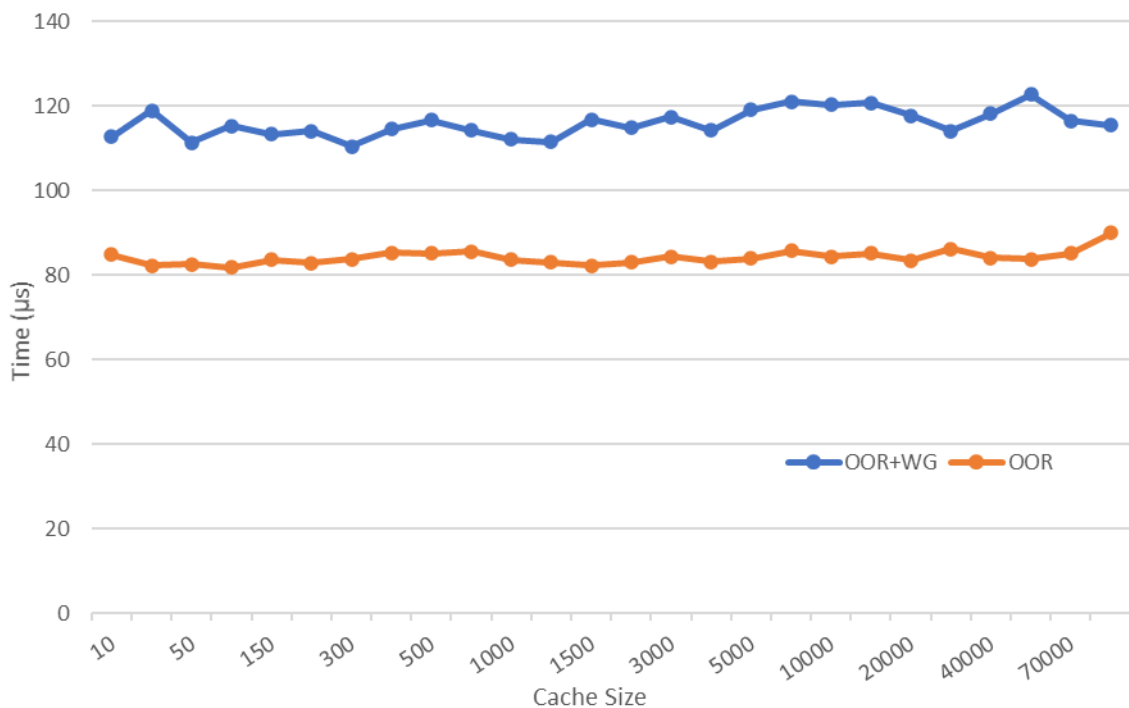


Figure 4.4: Map Server response to Map-Request with different cache sizes.

As it is shown in Figure 4.4, the OOR+WG model takes a little bit more time to respond a Map-Request message from an xTR. The difference is negligible because we are talking about 40 μs . This difference is due to now the MS has to save also the public key associated with the EID in the cache. But, we don't lose so much MS response time including Wireguard what is a good point. Moreover, the cache size doesn't affect to the response time as we can see by the linear graphic behaviour, this is a great result for our model because we can have as many entries as we want in the MS cache without any losses in response time.

4.4. End-to-end Delay

When we talk about E2E delay, we refer to the time since we have one packet to send until this packet arrives at its destination. This process includes all the control-plane time to establish the data-plane secure tunnels and the communication through the Wireguard data-plane.

To do it, we needed the fastest ping that we could generate, the reason is simple, LISP doesn't have any queue where the packets wait until the system knows how to route them. If there is no entry in the cache for the incoming packet, it will be dropped. That's why we needed a fast-rated ping, if we could have sent one ping per second, the measures would have been erroneous because LISP doesn't have a packet buffer, so if a packet could not be sent, it will be dropped. Therefore, sending a fast-rated ping provide us better information about the performance of the system. Therefore, we used the following ping command to obtain the results: `ping -i 0.0001 -c 100 <ip>`. With this command, we are sending 100 pings in intervals of 0.1 milliseconds between them.

The measured value was the time since we sent the first ICMP-Request until the first ICMP-Reply. Between them, as we have said before, we lost some ICMP-Requests but the rate is higher enough to obtain nearly real results. we repeated the experiment 100 times with both models to compare the final results.

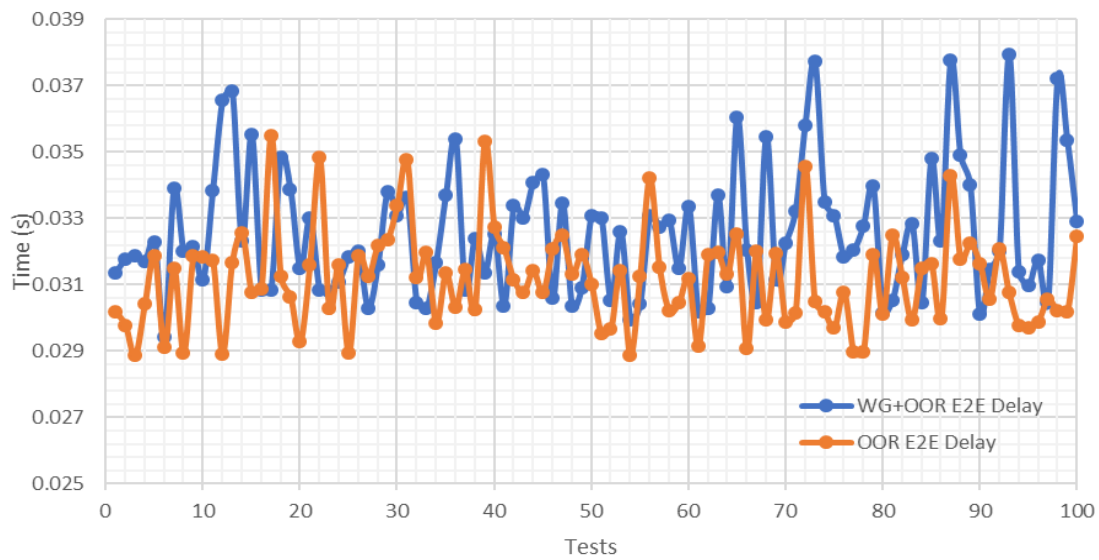


Figure 4.5: End-to-end delay graphic.

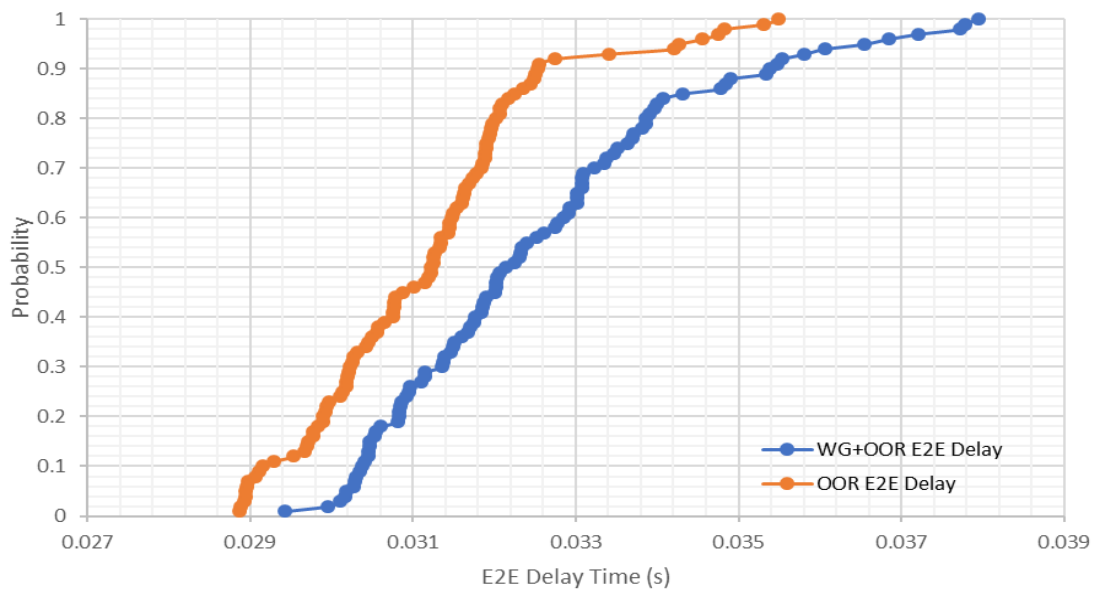


Figure 4.6: End-to-end delay CDF graphic.

In the presented results in Figure 4.5 and 4.6, we can see how the values are really close between WG+OOR and OOR. Cheeking the average value for both cases, we notice how close they are, in WG+OOR this value is 32.567179 and for OOR is 31.169138. We also can notice how in Figure 4.5 the values are not so stabilized, sometimes the OOR value is bigger than the merge of WG+OOR. However, if we look at the cumulative distribution function (CDF) plot, we clearly see how OOR presents lower values for the E2E delay.

The increment reason is the establishment of Wireguard data-plane tunnels. But, the difference is not so big, it would be interesting compare the relation security vs delay time. When we typically apply some kind of security into a protocol or whatever, there is a time increment due to the key generation, key management, key prove, etc. In the end, the cost of adding security into OOR is a limited delay that we expect from any cryptographic protocol.

4.5. Handover

As we have said before, to measure the handover value for both cases we changed the scenario. Until now, the scenario was three machines, two MN and one MS in the same network connected through Gigabit Ethernet. However, if we want to produce a handover in these conditions, we would have needed to change the RLOC in the MN by changing the IP manually or connecting it to another network, which is difficult and tricky at the end. Therefore, we decided to use the WIFI interfaces in the MN and install two AP to provide two different IPs. Then, the MN had two wireless interfaces which were manually configured to provide two different static IPs. So, to make a handover we started the OOR program sending packets from one MN to the other one, then in the middle of this communication we changed the wireless interface, the OOR software and Wireguard are prepared to deal and manage with the new RLOC IP, they were in charge of all the handover process. We used a continuous ping command: `ping -i 0.0001 <ip>` to establish a communication channel and evaluated 20 times the results for both cases.

This experiment consists in measure how long OOR and WG+OOR needs to deal with the RLOC change and re-establish the communication between the two MN. While we received ICMP-reply from the destination MN the communication was working, but when we changed the wireless interface, we had ICMP-requests without response until the handover was finished and the replies returned. We measured the handover time as the time between the first ICMP-Request without answer and the continuous ICMP-Request with his ICMP-Reply.

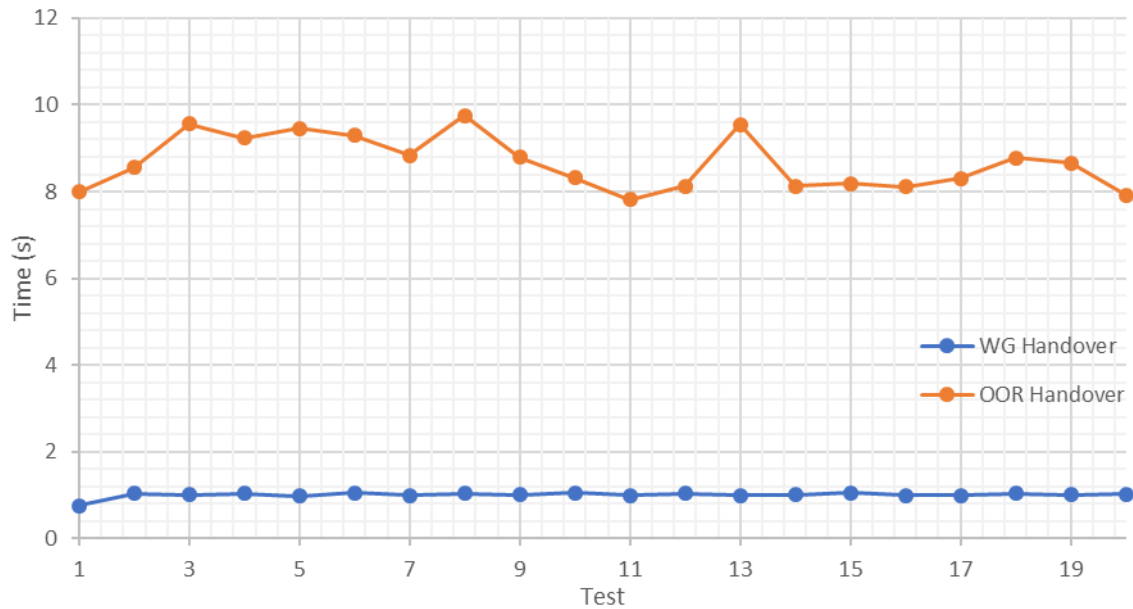


Figure 4.7: Handover time graphic.

The results in Figure 4.7 shows that WG+OOR manages the RLOC change faster than OOR. In the first case, the result is about 1 second, meanwhile, in the second case, the handover takes between 8 and 10 seconds. This improvement is thanks to Wireguard, it is made to manage the handover with high efficiency and how it is implemented in the Linux kernel. Also, changing the routes in the Wireguard interfaces is faster as well. That means, using Wireguard really improves our communication software in terms of dealing with RLOC changes.

To conclude with the experimental exposition, we make reference to all the BASH and PYTHON scripts used to obtain the presented data, which are in appendices 5. Each one is used for different intentions; thus, the code could change and be completely different between them.

5. Budget

Most of the cost has been the hours of work for both development and management, the hours of work of an engineer at € 9 / hour and those of a project manager at € 13 / hour have been calculated.

Then, we considered the Software and Hardware cost, where the main used programs were Eclipse C++, Excel, both of them are free license. Regarding hardware cost, we have used an ASUS GL533VD laptop with Windows 10 OS, a processor Intel® Core™ i7-7700HQ CPU @ 2.80GHz, 2808 Mhz, 4 main processors, 8 logic processors and 8.00 GB of RAM. Moreover, all the machines for the experimental scenarios, where we have 3 Linux 4.4.0 machines with 16G of RAM and 24 CPU working in 32 or 64 bits, and two access points (AP), LINKSYS WRT1200AC and NETGEAR WNDR3800. The cost of the machines is the amortized, not the total cost. The laptop price is around 2'000 € meanwhile the Linux Machines are around 4'000 € each one and finally, the Linksys AP is about 90 € and the Netgear around 180 €. We have used average values because the prices are variable.

Also, we have calculated the power consumption produced during the elaboration of the thesis, we have assumed the Spanish actual power cost which is 0.10329 €/kWh (IVA included). The laptop average consumption is around the 220Wh, for the Linux machines, we have considered a power consumption about 400Wh and for the APs an average of 350Wh. We used this value as an approximation, it can differ depending on different factors.

In the following tables, we are going to analyse the total thesis development cost, including the human and the software/hardware cost.

Human Resources			
Engineer	Hours	Price (€)	Total (€)
Investigation	244	8	1'952
Programming	120	8	960
Program Validation	24	8	192
Simulations and Tests	68	8	544
Results Evaluation	60	8	480
Thesis Writing	80	8	640

TOTAL	4'768 €
--------------	----------------

Table 5.1: Human Resources budget for Engineer work.

Human Resources			
Directors and Head Engineer	Hours	Price (€)	Total (€)
Support	80	13	1'040
Meetings	20	13	260
TOTAL			1'300 €

Table 5.2: Human Resources budget for the thesis directors.

Hardware and Software	Observations	Total (€)
Eclipse	Free License	0
Excel	Free License (OpenOffice)	0
Laptop	ASUS GL533VD (0.1 use rate)	200
Linux Machines x3	Linux 4.4.0 (0.1 use rate)	1'200
AP LINKSYS	WRT1200AC (0.1 use rate)	9
AP NETGEAR	WNDR3800 (0.1 use rate)	18
Power consumption	0.10329 €/kWh x 0.220kWh x 596h 0.10329 €/kWh x 0.400kWh x 68h 0.10329 €/kWh x 0.350kWh x 8h	16.64158
TOTAL		1'443.64158 €

Table 5.3: Budget for Software and Hardware.

Final Budget	7'511.64158 €
--------------	---------------

Table 5.4: Final budget considering all the proposed costs.

6. Conclusions and future development

Finally, with all the presented results and the developed mix between Wireguard and LISP, we have achieved what we proposed at the beginning of this thesis. We developed a completely functional and efficient control-plane for Wireguard that we can use to distribute the cryptographic material through all the users who need it, and also, we implemented a new security model to LISP with better results than the existing LISP-SEC protocol.

So, my thesis goal has been reached with great and encouraging results. Hence, in the future, this new model could be applied to different protocols constructing a completely different Internet security protocol stronger, easier and more advanced than the existing IPSec. Moreover, with the new LISP security implementation all the CISCO equipment (Routers, Switches, Hubs, Servers, etc) which nowadays supports this protocol could be updated with the security model increasing their value and making them more interesting for new possible customers/investors.

However, we developed a simple and primary model control-plane for such a protocol like Wireguard. There is still work to do in the future in order to implement a stronger control-plane which supports, for example, more Map-Servers to request the data, modify the model to support also multihoming, and even more using the developed control-plane to manage the handover through it. we don't have a strong and completely audited control-plane yet, we have made the first step forward, but now there is still work to do into this investigation field.

Bibliography

- [1] Farinacci, D. Fuller, V. Meyer, D. and D. Lewis, “*The Locator/ID Separation Protocol (LISP)*”, RFC 6830. DOI: 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.
- [2] Fuller, V. and D. Farinacci, “*Locator/ID Separation Protocol (LISP) Map-Server Interface*”, RFC 6833, DOI: 10.17487/RFC6833, January 2013, <<https://www.rfc-editor.org/info/rfc6833>>.
- [3] Saucez, D. Iannone, L. and O. Bonaventure, “*Locator/ID Separation Protocol (LISP) Threat Analysis*”, RFC 7835, DOI: 10.17487/RFC7835, April 2016, <<https://www.rfc-editor.org/info/rfc7835>>.
- [4] Farinacci, D. Meyer, D. and J. Snijders, “*LISP Canonical Address Format (LCAF)*”, RFC 8060, DOI: 10.17487/RFC8060, <<https://www.rfc-editor.org/info/rfc8060>>, February 2017.
- [5] Hu, Z. Zhu, Heidemann, J. Mankin, A. Wessels, D. and P. Hoffman, “*Specification for DNS over Transport Layer Security (TLS)*”, RFC 7858, DOI: 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [6] Farinacci, D. and B. Weis, “*Locator/ID Separation Protocol (LISP) Data-Plane Confidentiality*”, RFC 8061, DOI: 10.17487/RFC8061, February 2017, <<https://www.rfc-editor.org/info/rfc8061>>.
- [7] Fuller, V. Farinacci, D. Meyer, D. and D. Lewis, “*LISP Alternative Topology (LISP+ALT)*”, Work in Progress, draft-ietf-lisp-alt-10.txt, December 6, 2011.
- [8] Farinacci, D. and B. Weis, “*LISP Data-Plane Confidentiality*”, Work in Progress, draft-ietf-lisp-crypto-10, October 14, 2016.
- [9] Fuller, D. Lewis, D. Ermagan, Jain, A. and A. Smirnov, “*LISP Delegated Database Tree*”, Work in Progress, draft-ietf-lisp-ddt-09, January 18, 2017.
- [10] Maino, F. Ermagan, V. Cabellos, A. and D. Saucez, “*LISP-Security (LISP-SEC)*”, Work in Progress, draft-ietf-lisp-sec-17, November 29, 2018.
- [11] Maino, F. Kreeger, L. and U. Elzur, “*Generic Protocol Extension for VXLAN*”, Work in Progress, draft-ietf-nvo3-vxlan-gpe-07, April 10, 2019.
- [12] Cabellos, A. and D. Saucez, “*An Architectural Introduction to the Locator/ID Separation Protocol (LISP)*”, Work in Progress, draft-ietf-lisp-introduction-13.txt, April 02, 2015.
- [13] Jason A. Donenfeld, “*WireGuard: Next Generation Kernel Network Tunnel*”, in *Proceeding of the Network and Distributed System Security Symposium, NDSS 2017*, retrieved from <wireguard.com/papers/wireguard.pdf>, June 2018.
- [14] Rodriguez-Natal, A. Paillisse, A. Coras, F. Lopez-Bresco, A. Jakab, L. Portoles-Comeras, M. Natjaran, P. Ermagan, V. Meyer, D. Farinacci, D. Maino, F. and A. Cabellos-Aparicio, “*Programmable Overlays via OpenOverlayRouter*”, in *Proceedings of the IEEE Communications Magazine* (Volume: 55 , Issue: 6), June 2017.

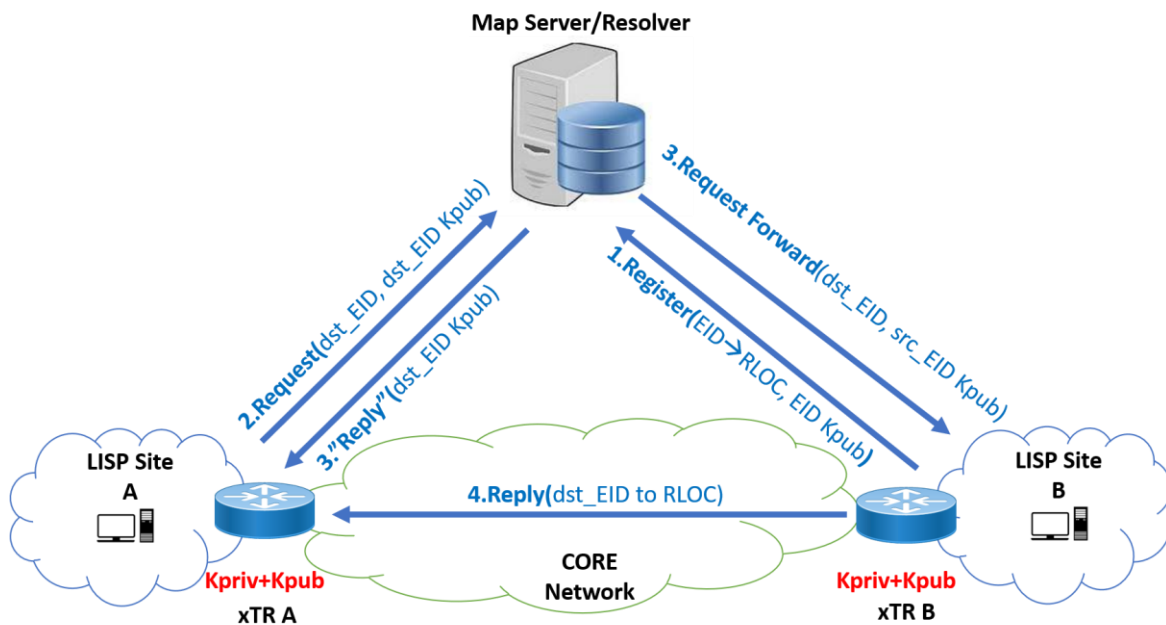
- [15] Jason A. Donenfeld, "Wireguard" [Online] Available: <https://www.wireguard.com/> [Accessed: 30 September 2019].
- [16] Wireguard (February 2020) [Online] Available: https://wiki.archlinux.org/index.php/WireGuard#Endpoint_with_changing_IP [Accessed: 23 November 2019]
- [17] Configuración de la VPN WireGuard en Ubuntu (February 2020) [Online] Available: <https://www.linode.com/docs/networking/vpn/set-up-wireguard-vpn-on-ubuntu/> [Accessed: 6 December 2019]
- [18] Locator/Identifier Separation Protocol (December 2019) [Online] Available: https://en.wikipedia.org/wiki/Locator/Identifier_Separation_Protocol [Accessed: 2 October 2019]
- [19] What is OOR? (February 2020) [Online] Available: <https://github.com/OpenOverlayRouter/oor/wiki> [Accessed: 2 October 2019]
- [20] Cisco Locator/ID Separation Protocol (LISP) [Online] Available: <https://www.cisco.com/c/en/us/products/ios-nx-os-software/locator-id-separation-protocol-lisp/index.html> [Accessed: 1 October 2019]

Appendices

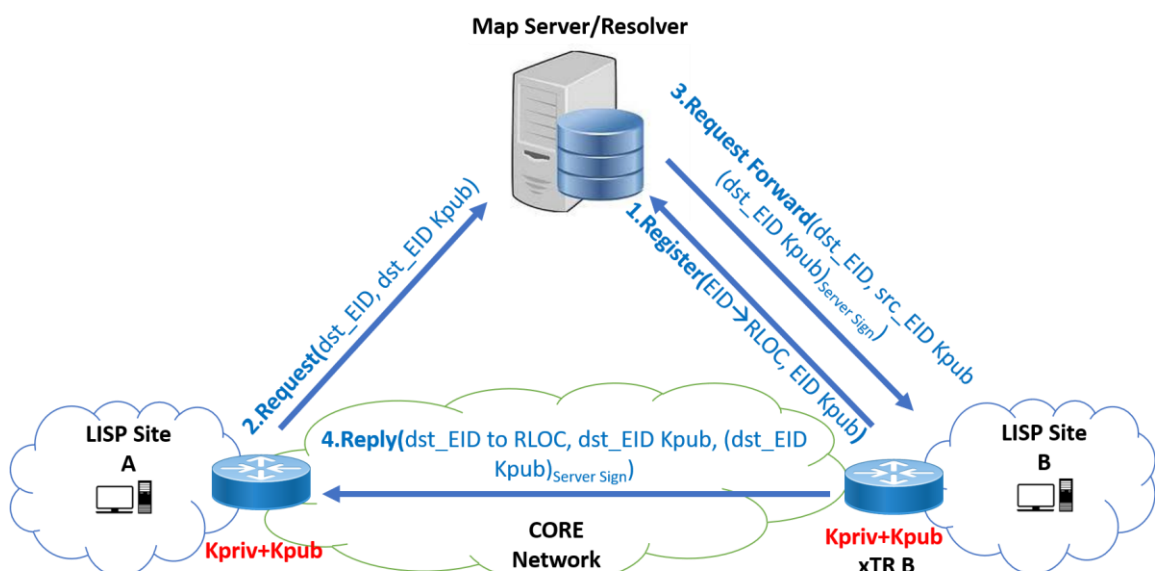
In this section we are going to add all the extended information of my thesis that we didn't present like graphs, diagrams, code, scripts, etc.

1. Data-plane key exchange

1.1 New Map-Server Message



1.2 Map-Server Signatures

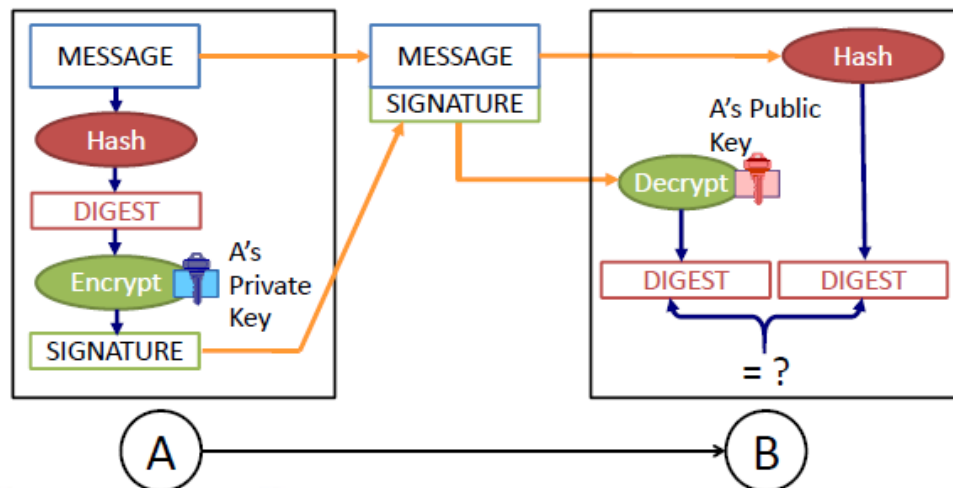


2. Digital Signatures

The signature and verification process follow the shown diagram, with this technique we provide data-authenticity and non-repudiation.

Digital Signatures

- Sign a secure digest of the message



3. Configuration File Changes

Here, we show the changes applied to the configuration file of the NM or xTR.

```
#####
#
# General configuration
#
# debug: Debug levels [0..3]
# map-request-retries: Additional Map-Requests to send per map cache miss
# log-file: Specifies log file used in daemon mode. If it is not specified,
# messages are written in syslog file
# ipv6-scope [GLOBAL|SITE]: Scope of the IPv6 address used for the locators. GLOBAL by default

debug                = 0
map-request-retries  = 2
log-file             = /var/log/oor.log
ipv6-scope           = SITE

# Define the type of LISP device LISPmob will operate as
#
# operating-mode can be any of:
# xTR, RTR, MN, MS, DDT, DDT-MR
#
operating-mode       = MN
```

This is the original mode, then, we added the device private key and control address to the main device configuration.

```
#####
#
# General configuration
#
# debug: Debug levels [0..3]
# map-request-retries: Additional Map-Requests to send per map cache miss
# log-file: Specifies log file used in daemon mode. If it is not specified,
# messages are written in syslog file
# ipv6-scope [GLOBAL|SITE]: Scope of the IPv6 address used for the locators. GLOBAL by default

debug                = 0
map-request-retries  = 2
log-file             = /var/log/oor.log
ipv6-scope           = SITE
dev-private-key      = "eGAt7SgAycLkq5U401B6R1T9WFOrNA+ZmNsb3ChY+20="
control-address      = 100.100.0.1

# Define the type of LISP device LISPmob will operate as
#
# operating-mode can be any of:
# xTR, RTR, MN, MS, DDT, DDT-MR
#
operating-mode       = MN
```

We also needed to change the MS configuration in this file.

```
map-server {
    address          = 192.168.1.190
    key-type         = 1
    key              = contraseña
    proxy-reply      = on
}

map-server {
    address          = 192.168.1.190
    control-address  = 100.100.1.1
    ms-public-key    = "t2XaPc0hfCQJFau+EmgBmh08qHgQMizDLNZNAEk61CU="
    key-type         = 1
    key              = contraseña
    proxy-reply      = on
}
```

Finally, the EID mapping.

```
database-mapping {
    eid-prefix       = 10.0.0.2/32
    iid              = 0
    ttl              = 10
    rloc-iface{
        interface    = eth0
        ip_version   = 4
        priority     = 1
        weight       = 100
    }
}

database-mapping {
    eid-prefix       = 10.0.0.2/32
    eid-prefix-kpriv = "eBf5LYdGyggYg6HlCl7VAAZwGMUIPpM/G3eF2FwWhF0="
    eid-prefix-kpub  = "WmB0zWYZDdGRinjTWNljaGbcixwbWb1BQ8zCHdmlzg="
    iid              = 0
    ttl              = 10
    rloc-iface{
        interface    = eth0
        ip_version   = 4
        priority     = 1
        weight       = 100
    }
}
```

Now, we are going to see the applied changes in the MS configuration file.

```
# Wireguard Configuration, define all the control-addresses from the xTRs
# and also their public keys. We can define several wisp_addr_t * lisp_addr_get_addr(lisp_addr_t *addr);ireguard-tunnels.
wireguard-peer {
    control-ip      = 100.100.0.1/32
    public-key      = "B1LMiFAjyIldLMak/bXripv2ETgzWPBMQ1beyVxZ0XY="
}

wireguard-peer {
    control-ip      = 100.100.0.2/32
    public-key      = "v+x60j2bw4temqUmQh+GmG50fdQS5s/+jMKQqx0xnlg="
}
```

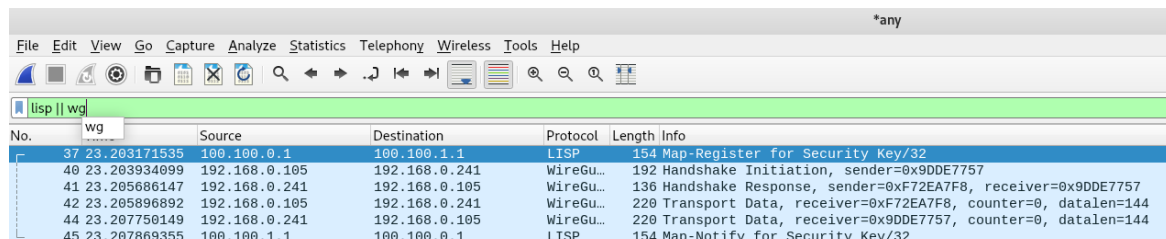
We added the xTR or MN peer in the configuration file, hence, the control-plane could be configured directly.

If we need to add manually one site, we need to include the site public key in this file too.

```
ms-static-registered-site {
    eid-prefix      = 10.0.1.32/24
    eid-prefix-kpub  = "WmB0zWYDdGRinjTWNljaGbcixwbWb1BQ8zCHdmLzg="
    iid            = 0
    rloc-address {
        address      = 100.100.0.1
        priority     = 1
        weight       = 100
    }
}
```

4. Wireshark encrypted LISP packets with Wireguard

We can see how the LISP packets go through the Wireguard interface without encryption (security) and then, they go out through the eth0 encrypted to the network.



No.	Time	Source	Destination	Protocol	Length	Info
37	23.203171535	100.100.0.1	100.100.1.1	LISP	154	Map-Register for Security Key/32
40	23.203934099	192.168.0.105	192.168.0.241	WireGu...	192	Handshake Initiation, sender=0x9DDE7757
41	23.205686147	192.168.0.241	192.168.0.105	WireGu...	136	Handshake Response, sender=0xF72EA7F8, receiver=0x9DDE7757
42	23.205896092	192.168.0.105	192.168.0.241	WireGu...	220	Transport Data, receiver=0xF72EA7F8, counter=0, datalen=144
44	23.207759149	192.168.0.241	192.168.0.105	WireGu...	220	Transport Data, receiver=0x9DDE7757, counter=0, datalen=144
45	23.207869355	100.100.1.1	100.100.0.1	LISP	154	Map-Notify for Security Key/32

5. Scripts

Here we will show all the used scripts for testing and simulating and we are going to see which is his particular function.

The first one is a simple script to create files and change the privileges.

```
#!/bin/bash

for i in {1..100}; do
    touch e2e_test_$i
    chmod o=rw e2e_test_$i
done
```

Then, we have one used to develop the E2E delay simulation, we used to get 100 test repetitions, execute/kill OOR and Tshark and execute a fast ping automatically.


```
#!/bin/bash

for i in {1..100};
do
    echo "Executing OOR..."
    /home/abarcia/oor/oor/./oor
    for a in {1..10};
    do
        echo $a
        sleep 1
    done
    echo "Executing Tshark with ping..."
    tshark -i any -f "icmp" -w e2e_/home/abarcia/e2e_test_results/e2e_test_${i}&
    ping -i 0.0001 -c 100 11.0.0.2
    echo "Killing Process Tshark&OOR..."
    killall tshark
    killall oor
    sleep 2
done
```

Another one to create the Wireguard keys faster.

```
#!/bin/bash

wg genkey | tee xtr_privatekey | wg pubkey > xtr_publickey
wg genkey | tee eid_privatekey | wg pubkey > eid_publickey
```

One to execute a Map-Request 50 times 1 per second using LIG.

```
#!/bin/bash

echo "We are going to send 50 requests, lps"

count=0
time=1
while [[ $count -lt 50 ]];
do
    echo `/root/Desktop/lig-lispmob-master/./lig -b -d -e -m 100.100.1.1 1.0.128.2`
    ((count++))
    sleep $time
done

echo "Script Done"
```

Additionally, one to create the MS configuration with different cache sizes, we needed to define how many sizes we wanted to define and then repeat the script for each one obtaining the complete configuration file fast and automatically.

```
#!/bin/bash

Entries=('10' '20' '50' '100' '150' '200' '300' '400' '500' '700' '1000' '1250' '1500' '2000' '3000'
'4000' '5000' '7000' '10000' '15000' '20000' '30000' '40000' '50000' '70000' '100000')
#Entries=('150000')

filename="/root/oor_files/BGP_PREFIXES_IPv4"

for i in "${Entries[@]}; do

if [ -e $i ]; then
    echo "File oor_ms_$i.conf already exists!"
else
    touch /root/oor_files/test_conf_files_keys/oor_ms_$i.conf
    echo "Creating file oor_ms_$i.conf"
fi

echo "#####"
#
# General configuration
#
# debug: Debug levels [0..3]
# map-request-retries: Additional Map-Requests to send per map cache miss
# log-file: Specifies log file used in daemon mode. If it is not specified,
# messages are written in syslog file
# ipv6-scope [GLOBAL|SITE]: Scope of the IPv6 address used for the locators. GLOBAL by default

debug                = 0
map-request-retries  = 2
log-file             = /var/log/oor.log
ipv6-scope           = SITE" >>/root/oor_files/test_conf_files_keys/oor_ms_$i.conf

#echo -n "Write the MS Private key: "
#read ms_priv
#echo "dev-private-key          = \"$ms_priv\" >>$1
echo "dev-private-key          = \"\"+08y7AfRXqSdVhNlddUIyiD9aEGAEQDPiU06o6u0EWQ=\"\" >>/root/oor_files/
test_conf_files_keys/oor_ms_$i.conf
```

To produce a ping with different IPs using a file, we created another script which sends a fast ping to 100 different IPs.

```
#!/bin/bash

filename="/root/oor_files/BGP_PREFIXES_IPv4"

count=1
ip="a"
while [[ $count -le 100 ]]
do
    echo $count
    read -r line
    IFS="/" read -ra ARR <<< $line
    if [[ "${ARR[1]}" != "24" ]] && [[ "${ARR[1]}" != "16" ]]; then
        echo "Change ip"
    else
        ip="${ARR[0]}"
        ping -f -c 100 -i 0.0001 $ip
        ((count++))
    fi
done < "$filename"

echo "Script Done"
```

Then, we arrive at the reading data scripts, to read some of the obtained data in the test/simulation we wrote scripts which separate the useful data. For example, the following script used to read the ICMP-Request and ICMP-Replies and obtain the E2E delay.

```
#!/bin/bash

touch /root/oor_files/e2e_latency_results/Complete_Results_00R

Delay_Time=()
for i in {1..100};
do
tshark -r /root/oor_files/e2e_test_oor_results/e2e_oor_test_$i > /root/removetext
filename="/root/removetext"
Times_Request=()
Times_Reply=()
while read -r line
do
if [[ "$line" == *"Echo (ping) request"* ]]; then
read -ra ARR <<< "$line"
Times_Request+=("${ARR[1]}")
elif [[ "$line" == *"Echo (ping) reply"* ]]; then
read -ra ARR1 <<< "$line"
Times_Reply+=("${ARR1[1]}")
break
fi
done < "$filename"

f_request=${Times_Request[0]}
f_reply=${Times_Reply[0]}

d_time=`echo "$f_reply - $f_request" | bc -l`
echo $d_time
echo $d_time >> /root/oor_files/e2e_latency_results/Complete_Results_00R
Delay_Time+=("$d_time")
rm /root/removetext
done

echo "Script Done!"
```

Or this one which was used to read the time difference between a Map-Request and Map-Reply of LISP protocol.

```
#!/bin/bash
IFS=" "

tshark -r $1 > /root/help.txt
#cat help.txt
touch /root/oor_files/results_map_cache_response/latency_results_master/$2

Times_Request=()
Times_Reply=()
Latency=()
count=0
filename="/root/help.txt"
while read -r line
do
do
wire=false
if [[ "$line" == *"Encapsulated Map-Request"* ]]; then
echo "ECM found"
read -ra ARR <<< "$line"
Times_Request+=("${Times_Request[@]}" "${ARR[1]}")
echo ${ARR[1]}
elif [[ $line == *"Map-Reply"* ]]; then
echo "REPLY found"
read -ra ARR1 <<< "$line"
Times_Reply+=("${Times_Reply[@]}" "${ARR1[1]}")
echo ${ARR1[1]}
elif [[ $line != *"Encapsulated Map-Request"* ]] || [[ $line != *"Map-Reply"* ]]; then
wire=true
fi

items_request=${#Times_Request[@]}
items_reply=${#Times_Reply[@]}
if [[ $items_request -eq $items_reply ]] && [[ $items_request -gt 0 ]] && [[ $items_reply -gt 0 ]] && [[ $wire = false ]]; then
delay=`echo -e "${Times_Reply[$count]}-${Times_Request[$count]}\n" | bc -l`
echo -e "$delay">>/root/oor_files/results_map_cache_response/latency_results_master/$2
Latency+=("${Latency[@]}" "$delay")
((count++))
fi
done < "$filename"
```

Glossary

IP (Internet Protocol)
OOR (Open Overlay Router)
LISP (Locator/Identifier Separation Protocol)
xTR (Ingress/Egress Tunnel Router)
EID (Endpoint ID)
RLOC (Routing Locator)
MN (Mobile Node)
RTR (Re-Encapsulating Tunnel Routers)
MS (Map Server)
UDP (User Datagram Protocol)
ICMP (Internet Control Message Protocol)
OS (Operation System)
P2P (Point to Point)
VPN (Virtual Private Network)
WG (Wireguard)
RFC (Request for Comments)
AP (Access Point)
IETF (Internet Engineering Task Force)
GPS (Global Position System)
BGP (Border Gateway Protocol)
UPC (Universitat Politècnica de Catalunya)
VXLAN (Virtual Extensible Local Area Network)
YANG (Yet Another Next Generation)
SDN (Software Defined Networks)
SFC (Service Function Chaining)
SSH (Secure Shell)
CRT (Cryptokey Routing Table)
LCAF (LISP Canonical Address Format)