Quantum Key Distribution in OpenSSL

Master of Science in Technology
Thesis
University of Turku
Department of Computing
Security of Networked Systems
November 2021
Aurora Papotti

Supervisors:
Petri Sainio
Seppo Virtanen

UNIVERSITY OF TURKU
Department of Computing

AURORA PAPOTTI: Quantum Key Distribution in OpenSSL

Master of Science in Technology Thesis, 53 p.
Security of Networked Systems
November 2021

Most of the current communications and systems rely on asymmetric cryptography, which is used to share a unique secret key between two parties communicating, in order to encrypt the information exchanged.

Recently, many researchers state that quantum computing will be a threat in 15-20 years. At the moment there is no quantum computer able to crack classical cryptography, however, a solution to address the threat should be found as soon as possible before classical cryptography reaches its expiration date, and all communications and systems will be cracked.

Quantum cryptography is considered a problem, but from another perspective, it is also the solution to it. In fact, this technology is strong enough to protect both from quantum and classical attacks. Quantum cryptography is considered secure because it is based on quantum physics laws.

The benefits of quantum cryptography, combined with the ones of symmetric cryptography offer an alternative solution to the Key Exchange problem: Quantum Key Distribution (QKD). The technology is a protocol that describes a cryptographic technique to exchange a secret key between two end users/applications within a communication.

This thesis starts by presenting the quantum threat, and the reasons that make quantum computing risky for classical communications and systems. Moreover, it states the importance to invest resources in this field of research in order to find a solution to address the problem once it will be a real risk.

Finally, I explain my contribution to Cefriel activities in the context of Quantum Key Distribution. The internship activity described is a demonstrative approach to integrate QKD technology into the OpenSSL library. The project aims to demonstrate the effectiveness and the feasibility of using QKD technology in SSL communications.

Keywords: Asymmetric Cryptography, Symmetric Cryptography, Quantum Cryptography, Quantum Key Distribution, TLS/SSL, OpenSSL

Contents

1 Introduction

Technology is a powerful tool that helped us in developing as a society. Many services, such as bank transaction, messaging services and online shopping, with the progress of the Internet became digitalised, and this evolution introduced many benefits but also it brought a new challenge: guarantee the secrecy of the information exchanged, and stored.

Over the last years, cryptography allowed us to protect ourselves from malicious attackers. Many services rely on a type of cryptography called asymmetric cryptography, which is used to share a secret key between two communicating parties that aim to exchange information secretly over an insecure channel. This kind of cryptography has always been enough to guarantee security, until the advent of Quantum Cryptography. The latter is based on quantum physics laws, and many researchers state that it may be a threat to our current cryptographic systems because it is more powerful. Quantum Cryptography is considered a huge problem, however, from a different perspective it is also the solution to it. In fact, quantum cryptography is secure against both "classical" attacks and quantum attacks.

In the context of cyber security one of the biggest challenge is the Key Agreement Problem: exchange a secret key between two users without being intercepted by an eavesdropper. Charles Bennett and Gilles Brassard took advantage of the quantum computing's benefits to propose Quantum Key Distribution: a protocol describing

a new and secure cryptographic solution to the problem of exchanging a secret key without being intercepted.

This thesis describes my contribution to Cefriel activities in the context of Quantum Key Distribution. Cefriel, with the collaboration of Italtel, the Polytechnic of Madrid, Telefonica, the Polytechnic of Milan, and CNR, participated in the Quantum-Secure Net[1] EIT Digital funded project in 2020. The objective of the project was to develop a simple and flexible solution for unconditionally insecure communication systems based on QKD technology, to make them work with actual metropolitan networks based on optical fiber. Cefriel developed two prototypal scenarios to demonstrate the usage of QKD technology:

- One scenario describes the usage in Blockchain context
- The other scenario describes the usage in SSL context.

Both scenarios aim to demonstrate that QKD technology can be applied both to the finance market, and IoT (Internet of Things) or IIoT (Industrial IoT) communications. On a broader level, the project can be adapted to any services or technologies that need a symmetric key.

During my internship activity, I focused on the usage of Quantum Key Distribution in the scenario of SSL, the aim was to use a QKD key as a pre-shared key for the SSL protocol. Over the last 6 months, I implemented a dynamic library to extend the OpenSSL library and to add support for the QKD technology. The activity has been simulated through an ETSI 004 simulator implemented by the University of Madrid to retrieve a QKD key, which is used as a secret key between two parties communicating.

The thesis is divided into two parts. The first part of the thesis gives an insight into the current state of classical cryptography and quantum cryptography. In

[1] `https://www.eitdigital.eu/fileadmin/files/2020/factsheets/digital-tech/EIT-Digital-Factsheet-Q-Secure-net.pdf`

particular, I described the cryptographic techniques which most services depend upon to offer security, highlighting their weakness against quantum computing. Moreover, I state the reasons why we should invest money and resources in the research of quantum cryptography. In conclusion, the second part of the thesis describes my internship activity at Cefriel, therefore, the implementation and design choices of my solution to introduce QKD technology in the OpenSSL library, the limitations of the solution, and some ideas for future works.

1.1 Motivation

This thesis aims to inform the reader about the quantum cryptography threat. The current cryptographic techniques are considered weak against this new technology, at the moment there are only theoretical demonstrations, however, it is important to perform researches in order to be ready when the risk will become practical. Government, industry, and academia are investing resources in projects to find solutions to the problem of quantum computing. Among the different projects, we find two main technologies: Quantum Key Distribution and Post-Quantum Cryptography. This thesis focuses on the description of the first technology, in particular, it proposes a solution to demonstrate an approach to add Quantum Key Distribution support in the OpenSSL library. This solution does not aim to offer a go-to-market product, however, it shows the effectiveness of integrating the quantum technology into a widely used protocol such as TLS/SSL.

1.2 Contributions

This thesis brings contributions to Cefriel activities in the context of Quantum Key Distribution. Last year Cefriel participated in the Quantum-Secure Net EIT funded project, and they developed two prototypal scenarios to demonstrate possible

usages of QKD technology: (i) one scenario shows its usage with Blockchains, (ii) the other shows its usage in SSL communications. During my internship activity, I had the opportunity to focus on the latest scenario. In this thesis, I demonstrate the effectiveness of integrating Quantum Key Distribution into one of the most used open-source SSL implementations: the OpenSSL library.

This thesis cannot be considered a contribution to the OpenSSL library, and the OpenSSL community. The project here described involves the implementation of a dynamic library that is loaded during the configuration of the OpenSSL library, which means, it is an independent third party from the main library, the support of the library's community is not involved. I made this implementation choice in order not to modify the source code, and take advantage of the maintenance and update of the OpenSSL library from the OpenSSL maintainers.

Chapter 4 describes in detail the implementation and design choices made, explaining the reasons for each choice. Moreover, in this part of the thesis, I state the limitations of my work, giving possible ideas for future developments and progresses.

1.3 Overview

This thesis gives a background about classical cryptography and quantum cryptography, and it describes the problem of the current cryptographic techniques' weakness against the threat of quantum computing. In addition, a personal internship activity is described to demonstrate a possible solution to the problem. This work is organized as follow:

- Chapter 2 introduces some basic notions about classical cryptography and quantum cryptography. Moreover, it explains the quantum threat problem and the reasons we should invest in this research to find solutions in order to

be ready for possible future attacks.

- Chapter 3 describes the Quantum Key Distribution protocol. I decided not to focus on the implementation details of different protocols because I think it was unnecessary for the purposes of the thesis, a general overview is enough to understand the proposed work.

- Chapter 4 described my Internship activity at Cefriel. The reasons and objectives of the project are described to begin with. In addition a general background about TLS/SSL protocol, OpenSSL library, and ETSI APIs is provided, in order to understand the technologies involved in the project. In conclusion, I describe in detail the implementation choices made to develop the proposed solution.

- Chapter 5 concludes this work with some final remarks and personal insights. In addition, some ideas for possible future work are proposed.

2 The Quantum Threat

The first section of this chapter describes the current state of cryptography and the classical cryptographic techniques. Most of the current systems are based on asymmetric cryptography, therefore, a general overview about about this kind of cryptography is given with the aim of helping the reader to fully understand the context of this work.

Section 2.2 describes the birth of Quantum Cryptography with the purpose of introducing the problem presented in Section 2.3: the Quantum Threat. Moreover, Section 2.3 intends to warn the reader about the risks with quantum computing, highlighting the importance of investing money and resources to study this technology, and find appropriate solution to the problem.

Section 2.4 introduces two technologies that are research's subject to address the problem of quantum attacks. The two technologies described in this section are Post-Quantum Cryptography, and Quantum Key Distribution.

2.1 The current state of cryptography

Everyone uses Internet to send messages, perform online transactions, and buy online. All these tasks need a security mechanism in order to protect and keep secret the information exchanged and stored for each user. When you open a new chat on a messaging service like WhatsApp, have you ever noticed the message *"This conversation is end-to-end encrypted"*?. This means that the conversation is

readable only for the two parties communicating, there are no third parties able to intercept the messages, and cryptography is accountable for making this happens.

Most of the current services are built upon cryptography to encrypt sensible information/data; in particular, they are based on a type of cryptography which is defined as **public-key cryptography**. The core function of public-key cryptography (or **asymmetric cryptography**), resides in the use of a pair of keys: a **public key** (which is is known to everyone), and a **private key** (which is known *only* to the owner's key). The generation of such keys is based on mathematical problems defined as one-way functions [1]. A *one-way function* is a function that is easy to compute given the input, but hard to compute in the opposite side given the output. An example of such problems is the *prime factorization*: it is easy to multiply two prime numbers, but it is really difficult to compute the decomposition of a number to find the prime numbers that compose it. If a person, or a computer, is able to solve these "hard" mathematical problems, then they are able to bypass the security mechanism. Most of the problems used in public-key cryptography are based on prime factorization (e.g. RSA), discrete logarithm (e.g. Diffie-Hellman), and elliptic curve (e.g. ECC). Although these algorithms are different from each other, they are based on a unique broader problem: the *hidden abelian group*.

In order to crack a pure cryptographic system[1] an attacker has two possibilities:

- Brute force attack: try all the possibilities to determine the right key that is able to decrypt the message. This kind of attack takes time, which usually depends on the key's length. A brute force attack is always successful, however, time is a big limitation for the attacker, which makes this attack inefficient.

- Finding the solution to the mathematical problem: this "attack" depends strongly on the robustness of the one-way function. In particular, we define

[1]With the term *pure* I mean a system based on modern (classical) cryptography.

unconditional computational security problems as those problems that are impossible to solve no matter the attacker's computational power. In contrast, **practical computational security** problems are quite impossible to solve with the current resources but may be easily breakable in the future [2].

2.2 Brief History of Quantum Cryptography

We have recently heard about Quantum Cryptography, and how it is slowly becoming a threat to our system. Many researchers have claimed that in the future (by now, not a too distant future) quantum computers will be able to exploit modern cryptography [3] (this aspect is described in Section 2.3). Firstly, I want to introduce quantum cryptography with a brief history of its origin.

Stephen Wiesner and Charles Bennett were two undergraduate students at Brandeis University in the early 1960', later the lives of the two students took different ways. Wiesner graduated from Columbia University, and Bennett from Harvard, despite this, they managed to keep in touch and meet regularly at the latter's communal house in Boston. One day Wiesner talked with Bennett about his idea: using quantum mechanisms in banknotes, making it impossible to counterfeit them according to the laws of nature. Wiesner's idea was based on the use of a *quantum multiplexing* channel, where one party could send two messages to the receiving party in a way that would allow the latter to decide which message to read, at the cost to destroy the other message irreversibly. The idea is described in the paper *Conjugate Coding* [4], Wiesner tried to submit it to the *IEEE Transactions on Information Theory*[2], unfortunately, the article was rejected [5].

One day in late October 1979, at the 20th IEEE Symposium on the Foundations of Computer Science, held in Puerto Rico, Charles Bennett mentioned

[2] `https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=18`

Wiesner's idea to Gilles Brassard. The latter was scheduled to give a talk on relativized cryptography, therefore, Bennett thought Brassard might be interested in Wiesner's findings [6]. The two researchers discovered how to incorporate the idea of Wiesner: *"The main breakthrough came when we realized that photons were never meant to store information, but rather to transmit it"* [7]. Bennett and Brassard published the first ever paper on quantum cryptography, the article was simply titled *Quantum Cryptography* and it was presented at Crypto '82 annual conference [8]. This publication raised interest in the original Wiesner's paper, which was finally published in 1983 on *SIGACT News*[3]. Finally in 1984, Bennett and Brassard, building upon their work, proposed BB84, a method for secure communication [5] [9].

2.3 Why quantum cryptography is a threat to classical cryptography

The classical cryptography based on asymmetric encryption has always been enough to protect systems from malicious attackers, but the evolution of quantum cryptography is endangering the entire modern cryptographic system. At the beginning, the quantum threat was considered only a theory, in fact, despite the quantum algorithm for integers factorization developed by Peter Shor [10], with the potential to decrypt RSA-encrypted communications [11], and several experimental progresses since the late 1990s, many researchers believed that *"fault-tolerant quantum computing is still a rather distant dream"* [12]. In contrast, over the last years investment in quantum computing research has increased [13], and as result, Google AI, in partnership with the U.S. National Aeronautics and Space Administration (NASA), on the 23rd October 2019, claimed to have

[3] `https://dl.acm.org/newsletter/sigact`

achieved *quantum supremacy* [14]. Moreover, in February 2019, IBM commercialized *IBM Q System One*, the first remotely usable quantum computer.

Quantum cryptography is based on quantum physics' laws. We all know the bit, the unit information in binary computing, which can assume the value 0 or 1, representing two distinct tension levels. Instead, in quantum computing the unit information is the **qubit**, whose value is a combination of 0 and 1, and may represent the spin of the electron, or the polarization of a single photon [15].

Quantum computers are faster and powerful, they are able to solve some kinds of problems more efficiently. According to some researches, the Shor algorithm, and other quantum algorithms, show that the time required to decrypt the keys used in asymmetric encryption increases slightly at the extension of cryptographic keys. A practical example could be the hidden abelian group, this problem can be solved in an exponential time with classical computing, instead, quantum computing is able to solve it in a polynomial time, respect to the key's length. All the algorithms that are considered practical computational security (e.g. RSA, ECC, AES) can be cracked in a time that is independent of the keys' length; the computing power and the time involved in the computation are *normal* [16] [17] [18].

At this point, we have to ask our-self: why do we need to invest in quantum research? Most of the researches show theoretical attacks, but the hypothesis that a quantum computer with enough power to exploit the modern cryptography will be created, is raising interest in this field of research. Moreover, while quantum cryptography is considered the threat on one side, it is also the weapon against it. In fact, quantum cryptography is secure against both classical and quantum attacks. However, there are some limitations with quantum cryptography: (i) both parties involved in a communication, need to have access to a quantum computer, (ii) that is quite expensive at the moment, therefore, it is inefficient and not very feasible [19]. Quantum cryptography is at an intermediary-advanced stage, there

are already some practical uses of it, and it has made a lot of progress over the years, however, its cost makes impracticable to absolutely replace all the modern cryptographic systems. Although the use of quantum cryptography in all contexts is something considered for a later future [20], there are several reasons why we should study quantum cryptography:

- Quantum cryptography is a new science and technology: companies, governments, and university are approaching it in different ways, therefore, it is not possible to have an accurate time estimation of when quantum computing will reach a level that will compromise the classical cryptographic systems. However, we have to be prepared once a powerful quantum computer will be created. We need to study, implement and test this new technology now to be reactive to the future threat.

- The migration from a modern system to a new one is always a tough process, in particular, when we are talking about cryptographic systems. Many years are required to reach the transition to a new technology, or simply an update of an algorithm. Moreover, many resources are involved: the entire infrastructure needs to be changed, the developers need to be trained, and old applications and new cryptographic standards need to be re-designed, not to mention the deployment of the new solution.

- Until now we have talked about the protection of information exchanged between two parties, but we should also be concerned about the protection of stored data. Companies are storing a huge amount of data, that is encrypted according to government legislation (e.g. GDPR). There are some data that we can consider irrelevant, but others contain sensitive information, and the main goal is to maintain their secrecy. This kind of data may include personal or health information (personally identifiable information/personal

healthcare information PII/PHI), or government information, therefore, a "lifetime" encryption is required.

2.4 Quantum threat solution

Currently, there is not a technology or a quantum computer that is capable of cracking our codes-practical, therefore, it is impossible to estimate an expiration date for the security of modern communications/systems. Nevertheless, many experts insist that the time to act and be prepared is now. In particular, the mathematician Michele Mosca of the Institute for Quantum Computing at the University of Waterloo in Canada declares that the chance that quantum threat will occur in 10 or 20 years is not a risk we can ignore, it is a threat to the global economy, and a defense mechanism should be planned as soon as possible [21].

Previously, in Section 2.1, I mentioned that most of the modern cryptographic systems on which we depend upon (e.g. to secure communications or to perform online transactions), rely on hard mathematical problems to solve, such as prime factorization, or discrete algorithms. These mathematical problems are the reason that makes modern cryptography enough strong to protect ourselves from cyber attacks. However, while these problems are impossible for today's computers to solve, it also leads to possible future risks. Imagine that an attacker is able to collect, and store sensible information encrypted with classical cryptographic techniques; if the attacker in 10 or 15 years manages to use a quantum computer, they will be able to decrypt the data.

Government, industry, academia are investing money and resources to prepare the world for this post-quantum era, in particular, Germany has invested €2 billion in quantum computing and related technologies over five years, funding over seven initiatives and incentivising collaboration between industry and academia [22].

The particularity of Quantum Cryptography is its dual role: (i) on one side is

considered a threat, (ii) on the other side it is the defense against both classical and quantum attacks. Therefore, if quantum computers are considered a problem (they can be used to break classical cryptography), they are also the solution for it (they can be used to build strong cryptography). In particular, the protocol Quantum Key Distribution was invented to address the problem of key agreement within a communication. Quantum Key Distribution is explained in Chapter 3.

There is another project run by the U.S. National Institute of Standards and Technology (NIST), that focuses on a different technology: Post-Quantum Cryptography [23]. In the next section, I present Post-Quantum Cryptography.

2.4.1 Post-Quantum Cryptography

The main characteristic of quantum cryptography is its power, in fact, it is based on quantum physics laws that make it unbreakable no matter how powerful the computer owned by an attacker is. There is no computing power, or algorithm that can crack this cryptography. Therefore, Quantum Cryptography is also being studied as a solution to deal with the future quantum threat. However, there are some side effects that make the use of Quantum Cryptography inefficient (at the moment). Both parties within a communication need to have access to a quantum computer, to begin with. In addition, the cost of this technology is still high, and it is not affordable for everyone: this makes quantum cryptography impracticable and inefficient.

Quantum Cryptography's side effects have prompted the research to focus on another type of cryptography: Post-Quantum Cryptography. This kind of cryptography can be performed with classical computers, but it is considered secure against attacks performed by quantum computers.

Section 2.3 describes public-key cryptography, which is based on mathematical problems that are hard to solve, however, it has been proven theoretically that

modern cryptography can be cracked with Shor's algorithm. Post-quantum cryptography can be considered as classical cryptography because it can be performed with classical computers, however, it is stronger than asymmetric cryptography because it is based on different mathematical principles that are difficult to solve even by a quantum computer. The terms quantum cryptography and post-quantum Cryptography can be easily misunderstood, but these two technologies have nothing in common; one needs a quantum computer to be performed, and the other does not.

Some reliable post-quantum ciphers against Shor's algorithm have been implemented [24] [25], however, post-quantum cryptography lacks the adjective unconditional computational security. It cannot be demonstrated that the post-quantum cryptographic algorithms are unbreakable in a polynomial time [26]. Like public-key cryptography, there is no mathematical proof of the effectiveness of this technology against quantum computers. Nevertheless, because of the difficulties with quantum cryptography as a solution to the quantum threat, it is interesting to investigate other types of solutions that can increase the security level of the modern communications and systems a bit. In addition, post-quantum cryptography fills the gap in the worst case scenario where the attacker is the only party that can have access to a quantum computer, and the victim cannot have access to it.

3 Quantum Key Distribution

This chapter begins with Section 3.1, which focuses on showing the benefits of symmetric cryptography combined with those of quantum computing to obtain an *optimal* cryptographic solution to the key agreement problem. The solution described in this chapter, to address the quantum threat, is the Quantum Key Distribution protocol, which is presented in Section 3.2. I decided not to describe the different implementations of Quantum Key Distribution because I used this technology as a "black-box" for my project. The main goal of my internship activity was just to incorporate this technology into SSL communications.

Section 3.3 presents several implementations of Quantum Key Distribution systems. These QKD networks have been developed by research institutes in different countries, in order to demonstrate the effectiveness of QKD technology. This section also aims to introduce some security issues that Quantum Key Distribution has to deal with. In conclusion, these challenges are described in Section 3.4.

3.1 Quantum computing and symmetric cryptography

Until now we have taken into account only asymmetric cryptography, the attacks we have mentioned do not concern symmetric cryptography (the same encryption

key is used by both sides). The theorem of Shannon developed in 1949 demonstrates that symmetric cryptography allows to obtain the perfect secrecy [27]. The benefits of quantum cryptography combined with those of symmetric cryptography, lead to a protocol for sharing an encryption key between two parties in a secure manner: Quantum Key Distribution. Before discussing this protocol, a brief recap of asymmetric cryptography and symmetric cryptography is provided below.

Public-key cryptography guarantees confidentiality. In a scenario where one party (Alice) wants to send a secret message to another (Bob), the workflow is as follows:

1. Alice encrypts a message with Bob's public key.
2. Alice sends the message to Bob.
3. Once Bob received the message, he uses his private key to decrypt the message.

The communication in this scenario is unidirectional, Alice is not able to read Bob's messages because she doesn't know his private key. Asymmetric cryptography is slower than symmetric cryptography, therefore, it is usually used to share a secret key between two parties: Alice sends a message to Bob which is the secret key, the latter is used to encrypt and decrypt all the communication between the two parties. In Figure 3.1 there is a schema that summarizes the functionality of asymmetric cryptography and symmetric cryptography.

The Key Establishment Mechanism (KEM), is the method by which cryptography keys are exchanged, which is also the main challenge in symmetric cryptography: exchanging the secret key between two parties without being intercepted. Therefore, asymmetric cryptography is used in the initial phase of communication to exchange the secret key without interference. A common scheme used to share the secret key was initially proposed by Diffie Hellman [28],

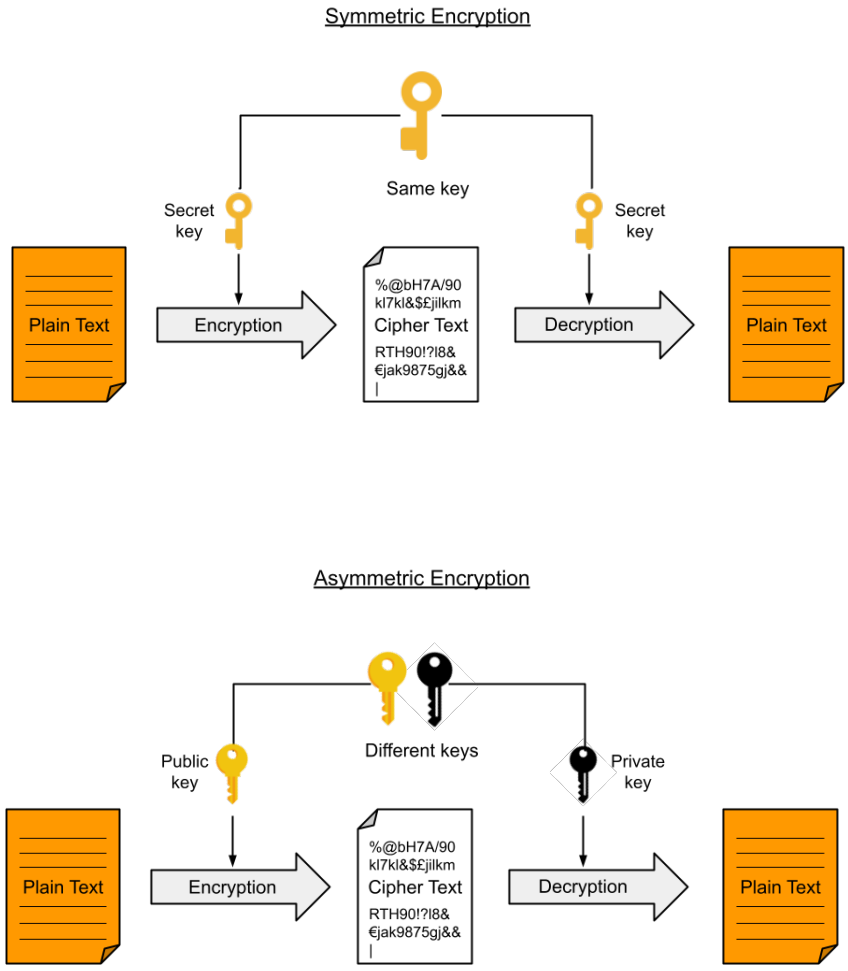

Figure 3.1: At the top of the figure, symmetric encryption with the same key shared between two parties. At the bottom, asymmetric encryption with two different keys: the private key and the public key. The public key is used for the encryption and the private key is used for the decryption.

now it is one of the foundations on which the SSL protocol is based.

3.2 What is Quantum Key Distribution

In Section 2.2 I briefly explained the story of quantum cryptography and the Quantum Key Distribution (QKD) protocol invented by Charles Bennett and Gilles Brassard in 1984. The protocol describes an alternative cryptographic solution to the key agreement problem: sharing a secret key between two parties without being intercepted. In contrast to public-key cryptography, QKD has been demonstrated to be unconditionally secure [29][30].

Quantum Key Distribution takes advantage of the quantum properties of photons to exchange a symmetric cryptographic key, which is used to encrypt messages exchanged over a "traditional" channel. The security of QKD resides in the universal natural laws that are reliable against any computing power, algorithm, or quantum computer. One principle that guarantees secrecy with quantum cryptography is a fundamental law of quantum cryptography: *"it is impossible to gain information about non-orthogonal quantum states without perturbing these states"* [31]. This means that the security in this kind of system resides in the event that an attacker tries to intercept the information exchanged. If an eavesdropper, commonly called Eve, tampers on the quantum channel connecting two legitimate users, Alice and Bob, she will leave traces of errors in the key exchange. In this case, Alice and Bob can decide whether to exchange a new key or interrupt the transmission.

Another benefit of QKD is related to information security, in fact, it is demonstrated to be an information-theoretically-secure system, which means that the system is unbreakable even if the attacker has unlimited computing power. Since the security of a QKD system does not rely on difficult mathematical problems to solve, the system is secure against both classical attacks and quantum attacks.

The last, but not least, property that QKD benefits from when it is used to generate several encryption keys in a row, is called forward-secrecy: all the different keys that are exchanged over a QKD link are independent of each other. Therefore, even if an attacker is able to compromise a key, he/she is not able to compromise all the others. This characteristic is highly appreciated both for guaranteeing a higher security level in networks, and for storing long-term data.

Basic principles of QKD

In this section, without going into details, I briefly describe the general structure and the basic principle of a QKD system: the **QKD link**. A QKD link is a point-to-point connection between two peers that want to exchange secret keys. A basic implementation of a QKD link consists of:

- **quantum channel**: an optical fiber channel capable of transmitting quantum information (qubit) between two users, Alice and Bob.
- **classical channel**: a classical communication channel that is public, but authenticated, between the two parties to perform the phases after the secret key is being exchanged.
- **exchange key protocol**: a protocol that takes advantage of quantum properties to ensure security by detecting interceptions or errors, and evaluating the amount of information lost or intercepted.

The workflow of a QKD system can be described as follows. Alice sends over the quantum channel a sequence of non-orthogonal quantum states of light encoded by a random stream of classical bits. Once Bob has received these quantum states, he performs some measurements and shares some classical data correlated with Alice's random stream. Then, the classical channel is used to test the correlations between Bob's data and Alice's data. High correlations statistically imply that no

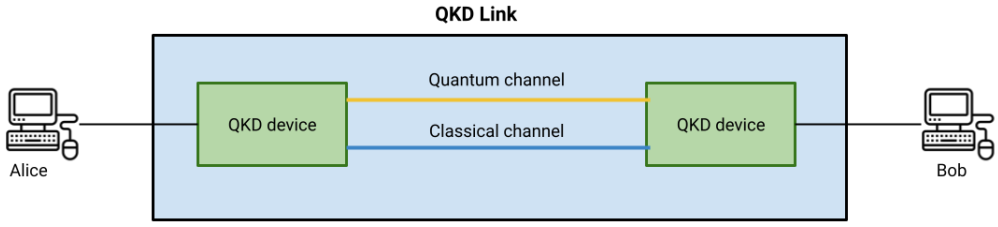

Figure 3.2: Schema of a basic QKD system. The QKD link is composed by a quantum channel, and a classical channel. Two users, Alice and Bob, communicate by means of the QKD link to share a secret key.

eavesdropping has taken place on the quantum channel. In the opposite case, it is necessary to abort the key generation process and start it again.

3.3 Quantum Key Distribution networks

Some research institutes over different countries invested enormous resources to implement Quantum Key Distribution systems, in order to demonstrate the effectiveness and feasibility of QKD technology. The first QKD network is the DARPA quantum network, which was proposed by BBN Technologies in collaboration with Harvard and Boston universities. DARPA quantum network has 10 quantum nodes and adopts a hybrid network type (i.e., active optical switch and trusted node networks) [32] [33].

In 2004, the project SECOQC QKD network was launched. This project defines practical applications of QKD networks with the aim of analyzing QKD networks' issues, in particular security aspects, communication protocols, design and architecture, and implementation methods [32] [34].

Another example is the Tokyo UQCC (Updating Quantum Cryptography and Communication) QKD testbed network launched in Japan in October 2010. The key

distribution service of this QKD network was used to perform a live demonstration of secure TV conferencing [32].

There are plenty of experiments and studies that provided beneficial results in terms of the network framework, key generation rate, communication distance, and routing protocol. However, Quantum Key Distribution presents some issues and challenges and issues that need to be addressed; in particular, these issues involve the security aspect. The most important security challenges that Quantum Key Distribution has to deal with are:

- the lack of a point-to-multipoint mechanism in QKD networks.
- the lack of a suitable security interface between the classical end users/applications and the quantum nodes.

3.4 The limits of Quantum Key Distribution

Quantum Key Distribution seems to perfectly guarantee the integrity of the keys, however, it does not mean that unhackable communications are within our reach.

Section 3.3 describes several Quantum Key Distribution networks that have been implemented over the last years. However, while QKD technology has many benefits, it has to deal with some security issues that are subjects of research. The first major issue is the lack of a point-to-multipoint mechanism in QKD networks, in fact, all the QKD networks allow two remote end users/applications to distribute session keys, providing a point-to-point key distribution service: there is no point-to-multipoint mechanism [32].

Another issue is the lack of an adequate security interface between the classical end users/applications and the quantum nodes. In Section 2.4, I mention the high construction cost of quantum technology, which makes it unfeasible for an end user/application to have access to a dedicated quantum node to access the service

implemented by a QKD network. Therefore, in order to have access to a quantum node, several end users/application need to share one quantum node, which means they still use a classical network to link a quantum node. It is important to design a security mechanism for the communication between the end users/applications and the quantum nodes [32].

Moreover, Quantum Key Distribution needs relays, which leads to another weakness. When two communicating parties are far apart, the QKD networks need repeaters to transmit messages, and these repeaters may be a hackable point. In addition, the use of routers and hubs is necessary in QKD networks to route messages, which makes them another weak point [35].

4 QKD in OpenSSL

Over the last 6 months, I contributed to Cefriel activities in the context of Quantum Key Distribution (QKD). Cefriel participated in the Quantum-Secure Net EIT Digital funded project in 2020. Its objective was the development of a prototype of a QKD transmitter/receiver. The project, whose activity leader was Italtel, included the Polytechnic of Madrid, Telefonica, the Polytechnic of Milan, Cefriel, and CNR. The QKD is a secure communication method that involves key distribution using quantum derivation techniques. Using this device, two entities that want to share a message and want it to be accessible only to them can exploit a particular type of key to ensure that no malicious third party has intercepted the message. QKD is the only known encryption method that, under certain conditions, and combined with a one-time pad, has unconditionally secured encryption symmetric key protocol and offers forward secrecy. The objective of my internship was to develop a complete software stack of an IPSEC/TLS over PSK library, adapting existing open-source SSL implementation. The activity is simulated through an ETSI 004 simulator.

This chapter begins with Section 4.1, which aims to give some basic notions of TLS/SSL protocol, providing the reader with an overview of SSL communications. Section 4.2 describes the OpenSSL library, the most popular open source implementation of the SSL and TLS protocols. The Section 4.3 briefly introduces the ETSI QKD APIs, then, how they should be implemented. These three sections

aim to explain to the reader the technologies involved in the project, explained in detail in the following sections of Chapter 4, in order to make easier the comprehension of the presented work.

In Section 4.4 I explain the general structure of the project, and the different approaches to introduce the QKD technology into the OpenSSL library. In particular, I present the motivation of my implementation choices. The Section 4.5 describes in detail the structure of the project, and which procedure I followed to implement it. In addition, Section 4.6 explains the workflow of the presented solution, providing a working demonstration.

Finally, Section 4.7 aims to present the challenges encountered during the development of this work, and the limitations of the final solution achieved.

4.1 TLS/SSL

Secure Sockets Layer protocol (SSL) was designed to provide secure communications over a computer network. Nowadays, SSL is deprecated and its successor Transport Layer Security (TLS) has replaced it. This protocol is mainly used in all situations where application layer information needs to be end-to-end encrypted before TCP transmission. For example, it is used as the security layer in HTTPS, and in applications such as email, instant messaging, and voice over IP.

TLS was first proposed by Internet Engineering Task Force (IETF) in 1999, now, the current version is TLS 1.3, and it was defined in August 2018. Before TLS, the SSL protocol was developed by Netscape Communications with the goal of adding the HTTPS protocol to their navigator web browser. Earlier SSL specifications were helpful in implementing the TLS protocol.

SSL is obsolete, but since it is the first implementation of securing the application layer messagges before transportation, and the later TLS is based on it, the two terms are used interchangeably. However, the two protocols are different, in particular, the

main difference relies on the level of security; TLS is an upgraded and more secure version of SSL, therefore, it is widely used throughout the Internet. A server can guarantee different versions of SSL and TLS, however, using the highest version of the protocol is recommended to avoid vulnerabilities related to older versions (e.g. SSL 3.0 or TLS 1.0). Today, when someone refers to the protocol with the term SSL they actually mean TLS, in order to be consistent, through the section I will refer to the subject protocol with the term TLS.

The purpose of the TLS protocol is to provide privacy and data integrity between two or more communicating parties (e.g. computer applications). The protocol runs on top of the transport layer in the TCP/IP model, and it consists of two layers: the TLS record protocol, and the TLS handshake protocol.

4.1.1 Workflow

The protocol is used within a communication across a network between a Client-Server application, the aim is to provide privacy and data integrity against tampering and eavesdropping. As I mentioned previously, the protocol itself is the composition of two layers: the TLS Record Protocol and the TLS Handshake protocol. At the lowest layer the TLS Record Protocol is implemented , and it offers a secure connection with two properties:

- Private connection: in order to guarantee secrecy, symmetric cryptography is used for data encryption (e.g. AES, RC4, etc.). The keys are based on a secret negotiated by the TLS Handshake Protocol, and they are unique for each different connection.

- Reliable connection: a keyed MAC is included in the message to provide data integrity. Secure hash functions (e.g. SHA-1, etc.) are used for MAC computations. Moreover, the Record Protocol can operate without encryption, differently, it cannot operate without a MAC.

The TLS Record Protocol is an encapsulation of the higher protocols, such as the TLS Handshake Protocol. The latter ensures authentication and negotiation of encryption algorithms and cryptographic keys between server and client, before the first byte of data is transmitted or received by the application protocol. The connection security provided by the Handshake Protocol has three properties:

- The peer's identity is authenticated by means of asymmetric (public key) cryptography (e.g. RSA, DSA, etc.). Authentication may be optional but is required at least for one of the peers.
- The shared secret negotiated between the client and the server is inaccessible to eavesdroppers. Even if an attacker who is able to place himself in the middle of the communication, he is not able to obtain the shared secret.
- There is no attacker who can modify the negotiation without being detected by the communicating parties. Therefore, the negotiation is reliable.

In the next two sections, I briefly explain the TLS Record Protocol and the TLS Handshake Protocol.

TLS Record protocol

Once the Record Protocol has received the data from the application layer, several operations are performed:

1. **Fragmentation**: the data is fragmented into blocks. A sequence number is added to each block to protect against attacks that attempt to reorder data.
2. **Compression**: the data is compressed according to the algorithm negotiated during the handshake phase.
3. **Add MAC**: a MAC is applied to the data to guarantee data integrity of the outgoing messages.

4. **Encryption**: the data is encrypted using the cryptographic keys and algorithms negotiated during the handshake phase.

5. **Append TLS Record Header**: a TLS Record Protocol header is applied to the data.

As final step, the data is sent to the TCP protocol of the transport layer for the transmission. In case the data is an incoming message, the reverse process in performed.

TLS Handshake protocol

The TLS Handshake Protocol's aim is to negotiate the security parameters (e.g. encryption keys and cryptographic algorithms) of a data transfer session. It consists of a series of sequential messages; the procedure (shown in Figure 4.1) describes a basic handshake with only the server authenticated (the client is not authenticated). The provided description refers to TLS 1.2 handshake.

1. **Client Hello** (client): the client starts the TLS handshake with a ClientHello message specifying:

   - the highest TLS protocol version supported by itself
   - a list of supported cipher suites and compression methods
   - a random number
   - a session ID in case a client attempts to perform a resumed handshake

2. **Server Hello** (server): the server replies with a ServerHello message specifying:

   - the chosen protocol version: it should be the highest that both client and server support.

- the chosen cipher suite and compression method from the list offered by the client.
- a random number
- a session ID to allow or confirm a resumed handshake.

3. **Server Certificates** (server): the server sends his digital certificate to prove its identity.

4. **Server Hello Done** (server): a ServerHelloDone message is sent by the server to indicate the negotiation handshake is done.

5. **Client Key Exchange Message** (client): this message sent by the client may contain a PreMaster secret, which is encrypted using the server's public key.

6. **Key Generation** (client/server): The client and the server use the random numbers and the PreMaster secret to generate a **master secret**. All the symmetric encryption keys, or session keys, used in this connection are obtained from the master secret.

7. **Cipher Spec. Exchange** (client): this message notifies the server that all the following messages will be encrypted using the negotiated keys and algorithms.

8. **Finished** (client): this is the first encrypted message, moreover, it contains a hash and MAC of the entire conversation. The server has to attempt to decrypt the message and verify the hash and MAC. If one of the two fails, the connection should be dropped.

9. **Cipher Spec. Exchange** (server): this message notifies the client that all the following messages will be encrypted using the negotiated keys and algorithms.

10. **Finished** (server): the server sends his Finished message. The client has to perform the same decryption and verification procedure that the server performed previously.

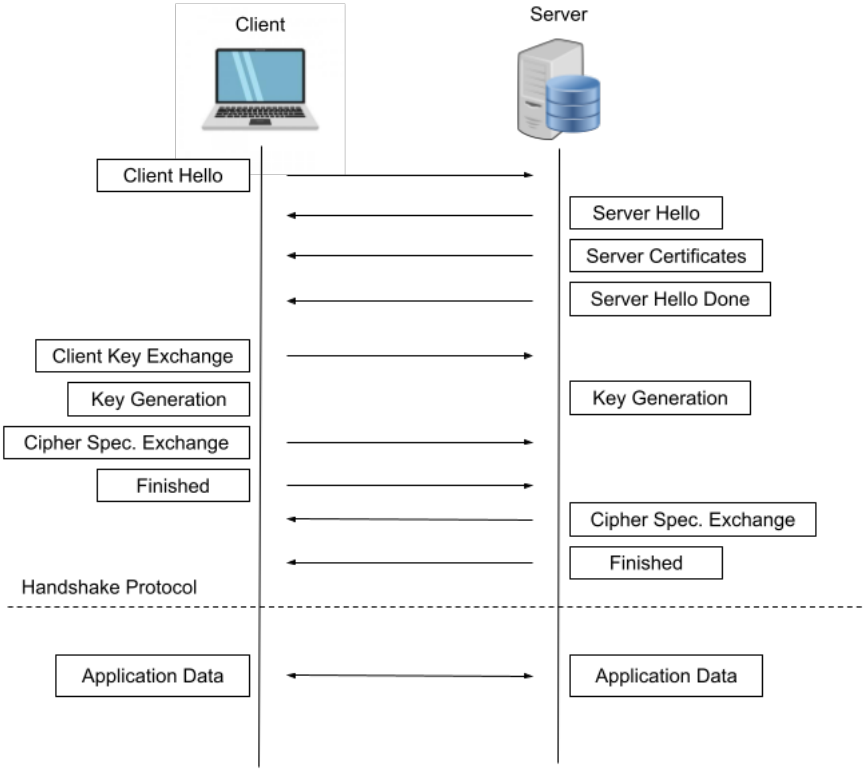

Figure 4.1: TLS Handshake Protocol phase

TLS 1.3 Handshake

Differently from TLS 1.2 handshake, there is only one round trip. The client sends a ClientHello message containing a list of supported ciphers according to client's preference order, moreover, it makes a guess on the key algorithm that will be used, in this way, it immediately shares a possible secret key. The server replies with a ServerHello message containing its key, a certificate, the chosen cipher and the finished message. Finally, once the client has received the server's Finished

message, it sends a Finished message too. Server and Client are both coordinated on which cipher suite to use.

4.2 OpenSSL

OpenSSL provides two tools: (i) a SSL toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols, (ii) and a general-purpose cryptography library. It is a software library widely used by Internet servers, including a lot of HTTPS websites. The library is written in the C programming language and is licensed under an Apache-style license, which means that users are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions.

The OpenSSL project, based on a fork of SSLeay by EriAndrew Young and Tim Hudson, and was founded in 1998, the founding members were Mark Cox, Ralf Engelschall, Stephen Henson, Ben Laurie, and Paul Sutton. Actually the OpenSSL management committee is composed by 7 people, and there are 17 developers.

4.3 The ETSI QKD API

The acronym ETSI QKD API stands for European Telecommunications Standards Organization[1] (ETSI) Quantum Key Distribution (QKD) Application Programming Interface (API). These APIs are necessary to interface from the QKD consumer (end application/user) to the QKD provider (the QKD device), and vice versa.

[1] `https://www.etsi.org/`

4.3.1 QKD Application Interface Specification Description

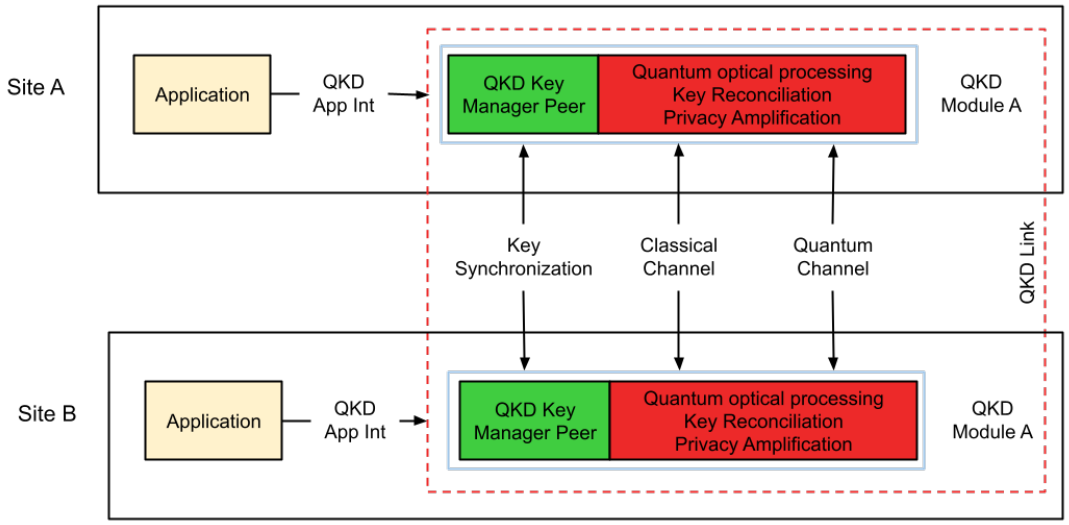

Figure 4.2: QKD Application Interface and peer relationships. Sites A and B represent security perimeters at each site (simple schema).

In Figure 4.2 there is a single QKD link enclosed by a red dashed box, the two endpoints reside at site A and site B. Each site includes a single application (the yellow box), and a single QKD Module enclosed by a blue box. The QKD Module implements the QKD protocol (the red box) used to produce QKD keys, that are managed by the QKD Key Manager peer (represented by the green box). In this case the QKD API is used by the single peer application to acquire identical sets of secure keys on demand [36].

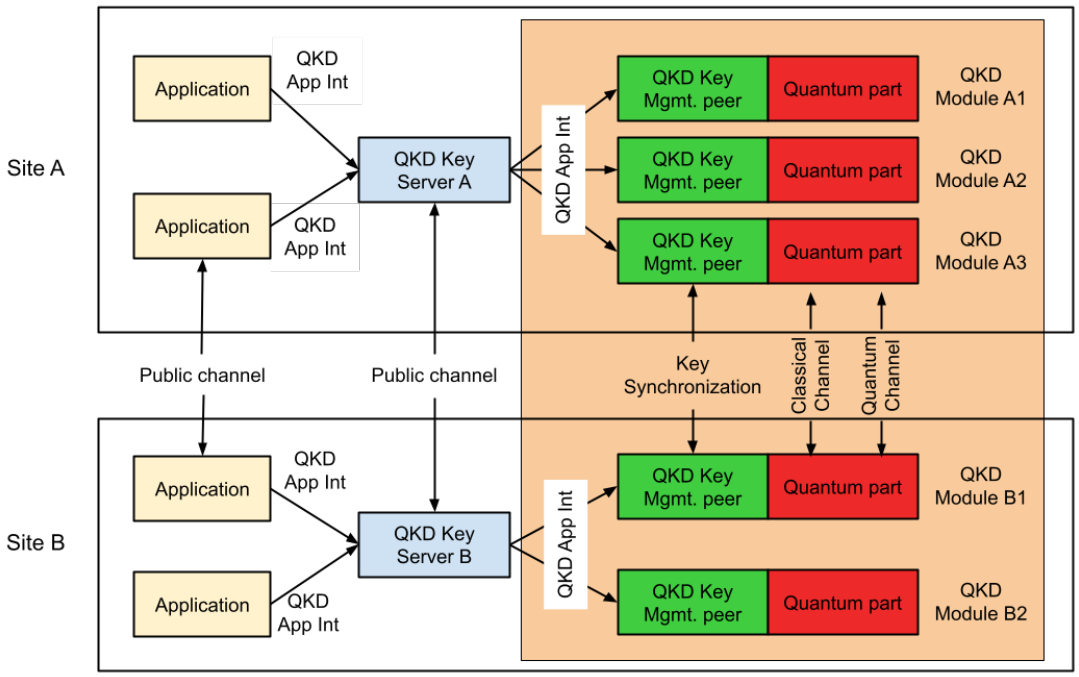

Figure 4.3: QKD Application Interface and peer relationships. Sites A and B represent security perimeters at each site (complex schema).

A more complex (and general) scheme is shown In Figure 4.3. There are two sites inside a network that contain two applications. A single QKD link formed by QKD Module A3 and QKD Module B1 connects Site A and Site B. The other QKD modules' endpoints are not shown in the figure. The QKD Key Servers (the blue boxes) represent a network layer Key Manager and their objectives are to manage keys between endpoints, and to deliver identical sets of keys to these endpoints for the peer applications. In this QKD API a secure key is guaranteed from the QKD Key Manager peer (link layer), to the QKD Key Server (network layer), as well as from the QKD Key Server to the applications [36].

4.3.2 QKD Application Interface API Specification

OPEN_CONNECT	A (Key_stream_ID) association is reserved for a set of future keys at both endpoints of the QKD link. Moreover, a set of parameters is established to define the expected levels of key service. `OPEN_CONNECT()` should be a blocking function, no further operations can be performed until both peers are connected or until the timeout value is exceeded.
CLOSE	This function terminates an association established for a certain Key_stream_ID. After, no more keys can be allocated for this Key_stream_ID. Since there is timing differences between the endpoints of the link, this operation will take place at another time, therefore, any unused key should be kept until that occurs and then discarded, or the time to live value of the QoS parameter is exceeded.
GET_KEY	This API is used to obtain the amount of key material requested for a specific `key_stream_ID`. The return value can be a `key_buffer` parameter containing the fixed amount of requested key, or an error message in case of failure. The `GET_KEY()` function may be called as often as desired, and the QKD key manager should reply at the bit rate specified in the QoS parameter, or at the best rate the system can manage.

Table 4.1: Brief description of the API functions that a QKD key manager has to implement as application interface

4.4 Implementing QKD in OpenSSL

In this section I explain in detail the implementation of the solution to introduce the QKD technology into the OpenSSL library, guaranteeing secure communications between two parties, through the TLS/SSL protocol. The peculiarity resides in the use of a secret key previously exchanged in a secure way thanks to the use of an ETSI emulator.

The scenario presented aims to demonstrate the effectiveness and feasibility of introducing QKD technology in a real context such as TLS/SSL protocol, which is the core of all secure communications.

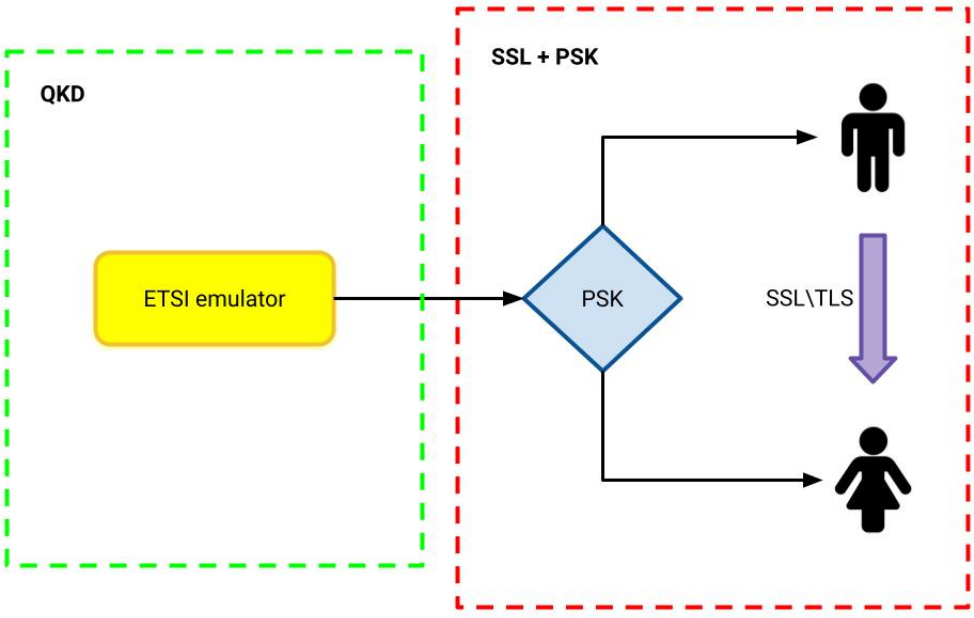

Figure 4.4: Schema of QKD technology integrated with TLS. The dashed green box is the ETSI emulator, and the dashed red box is the TLS component, which retrieves the QKD key from the emulator.

QKD

The role of the Quantum Key Distribution method is played by the ETSI 004 emulator implemented by the Universidad Politécnica de Madrid, which aims to provide a QKD key: the secret key shared between two users like Alice and Bob. The ETSI emulator has been downloaded in a local copy, and to execute the process to retrieve the QKD key, a python script is used to start the procedure (Figure 4.5).

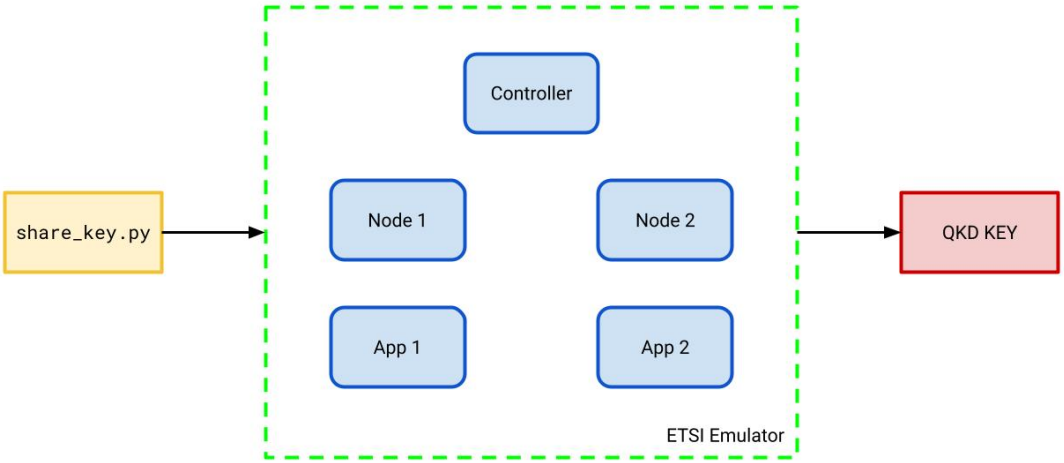

Figure 4.5: Schema of the QKD system. The python script `share_key.py` is responsible to run the ETSI 004 simulator, the output is the QKD key retrieved.

TLS/SSL with Pre-Shared Key

Once the ETSI emulator has finished its process, the obtained QKD key is passed to Alice and Bob. The shared secret is obtained before the TLS connections, and it is used by the two users to exchange messages over a channel, which is considered secure thanks to the advantages of QKD (Figure 4.6).

However, this work is not risk-free, and some clarifications are necessary. The QKD key retrieved from the ETSI emulator, and shared between Alice and Bob, is passed over a socket communication, which means that the transmission of the QKD key relies on classical cryptography, to begin with. This implementation choice lacks security, but it is enough for proof of concept purposes, it demonstrates the feasibility of introducing QKD technology into the OpenSSL library. Moreover, this work only describes a demonstrative approach; the goal is not to achieve a go-to-market solution.

Finally, due to lack of resources, Alice and Bob lie on the same virtual machine, therefore, the server used for the communications, and for SSL certificates, is the local host of the virtual machine. Of course, in a real scenario, Alice and Bob are physically in two different geographical places, therefore, it would be necessary to introduce a VPN tunnel between the two locations and above all create a server that allows communication with appropriate certificates. However, the implementation of the project in a wider context is out of scope for the purposes of this thesis, moreover, for time and resources reasons, it was not possible to develop it. The final solution achieved is good enough to demonstrate an approach to use QKD technology in TLS/SSL communications.

4.4.1 Approaches to add QKD support to OpenSSL

In order to extend the OpenSSL library to add support for the QKD technology in OpenSSL using the ETSI QKD API there are two different possibilities:

- **create an OpenSSL engine**: in this case we can simply "abuse" an existing classical protocol like Diffie-Hellman. Therefore the solution would be to hack the existing engine-based extension mechanism for Diffie-Hellman.
- **modify the OpenSSL state machine**: in this case we should introduce QKD as a new first-class key exchange protocol.

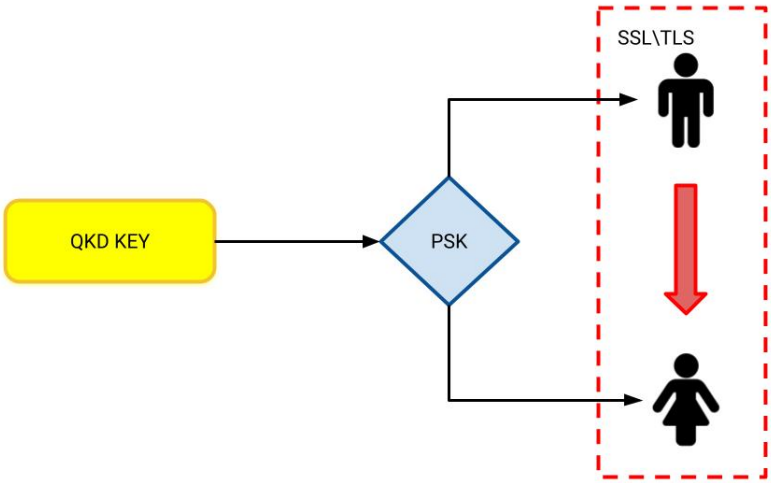

Figure 4.6: The QKD Key obtained by the ETSI emulator is used as symmetric encryption key in TLS/SSL.

Hacking existing engine-based extension mechanism for Diffie-Hellman

The OpenSSL engine mechanism allows third parties to extend the OpenSSL engine. The engine is an extension that can be implemented as a dynamic library (.dylib files on macOS or .so files on Linux), and it is loaded during the OpenSSL initialization without modifying the OpenSSL source code itself. OpenSSL configuration files are used to control which extensions should to be loaded into OpenSSL. This solution is simple, and since we do not make any changes to the source code, it allows to keep the OpenSSL library updated and maintained according to the official releases.

OpenSSL engines were created with the intent of offloading time-consuming cryptographic operations from the default software implementation in OpenSSL, using special-purpose crypto acceleration hardware instead. The library's maintainers have decided a-priori which operations should be offloaded. In order to offload these specific cryptographic operations, some APIs have been implemented.

The engine APIs allow a dynamically loaded engine to register a callback function that OpenSSL will call when it needs to be performed. This registered function is used instead of OpenSSL's default software implementation.

In my current implementation of QKD support in OpenSSL I decided to use the engine mechanism, in order to avoid modifying the OpenSSL source code (approach explained in the next section). However, engines for QKD protocols are not currently supported in OpenSSL, therefore, it is necessary to "hack" an existing classical protocol. In my case I decided to "abuse" of Diffie-Hellaman API (DH). The use of this approach avoids implementing the whole state machine for the engine: OpenSSL does it for you.

In order to overload the Diffie-Hellman protocol it is necessary to overload the proper callbacks in the `DH_METHOD` structure. In my case I overloaded two callback functions:

- The Diffie-Hellman `compute_key` engine callback: this one is called to perform the first step of Diffie-Hellman key exchange, which is choosing a private key and computing the corresponding public key. In my case I do not need the private and the public key, therefore I hacked this callback by simply defining a fixed pair of private and public keys.

- The Diffie-Hellman `generate_key` engine callback: this one is called to perform the second part of Diffie Hellman key exchange, which is generating the shared secret using the pairs private-public keys of the parties involved in the communication, and the negotiated Diffie-Hellman parameters (g and p parameters). I hacked this callback to instead open a socket communication with the QKD app of the ETSI simulator to retrieve the QKD key.

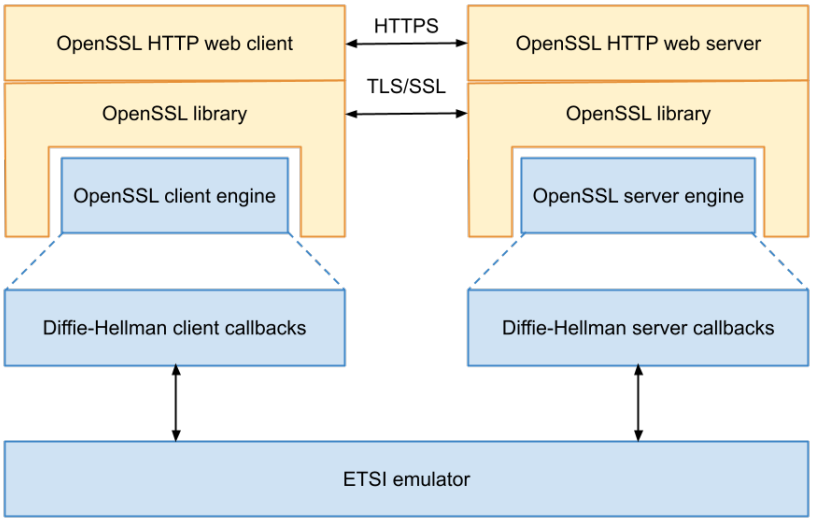

Figure 4.7: Schema of the implemented OpenSSL engine to introduce QKD technology into OpenSSL

Introduce QKD as a new first-class key exchange protocol

OpenSSL engines can only be used to accelerate a pre-determinated set of operations in existing cryptographic algorithms. As a result of how engines are implemented in OpenSSL it is not possible to use them to introduce a completely new key exchange algorithms such as QKD.

The way I have adopted to introduce QKD support does not affect the OpenSSL source code (summarized in the previous section), however, this solution cannot be consider a go-to market solution. The proper way to introduce QKD into OpenSSL would be to modify the source code, introducing QKD as a first-class abstraction in OpenSSL. An engine for QKD (not to hack Diffie-Hellman) would still be necessary because usually the QKD provider is implemented on an external device reachable through the ETSI API.

4.5 Hacking the OpenSSL Diffie-Hellman engine to add QKD

In this section I explain in detail how I hacked the existing OpenSSL Diffie-Hellman engine. In order to hack the engine it is necessary to overload the two callback functions of the `DH_METHOD` structure: `compute_key` engine callback, and `generate_key` engine callback. The library OpenSSL is huge, and it is very difficult to understand which callback functions should be overloaded. Luckily, while I was doing research on extending the OpenSSL library, I found the website of the Pan-European Quantum Internet Hackathon held on November 5-6th 2019[2], and organized by RIPE labs. A github repository with a mock implementation of the ETSI API, and a description of approaches to extend the OpenSSL library, was offered for the challenge. From their guidelines I considered the advise to "abuse" the existing engine-based extension mechanism for Diffie-Hellman. Moreover, in order to understand the how engines work in OpenSSL, and to have a general insight into the library, I used the following resources:

- OpenSSL wiki main page[3]
- OpenSSL wiki libcrypto main page[4]
- OpenSSL wiki Diffie-Hellman[5]
- OpenSSL wiki example ECDH engine[6]

[2] `https://labs.ripe.net/author/ulka_athale_1/take-part-in-pan-european-quantum-internet-hackathon/`

[3] `https://wiki.openssl.org/index.php/Main_Page`

[4] `https://wiki.openssl.org/index.php/Libcrypto_API`

[5] `https://wiki.openssl.org/index.php/Diffie_Hellman`

[6] `https://wiki.openssl.org/index.php/Creating_an_OpenSSL_Engine_to_use_indigenous_ECDH_ECDSA_and_HASH_Algorithms`

- OpenSSL wiki SSL and TLS tutorial[7]
- OpenSSL man page for engines[8]
- OpenSSL man page for DH_generate_key engine callback[9]
- OpenSSL man page for DH_compute_key engine callback[10]
- Gost-engine/engine GitHub repo containing an OpenSSL engine implementation[11]

The implementation of the OpenSSL engine to add QKD is structured on three different files:

- `qkd_engine_client.c`: this file contains the engine code that is unique to the client. It includes the implementation of the callback functions `compute_key` and `generate_key`
- `qkd_engine_server.c`: this file contains the engine code that is unique to the server. It includes the implementation of the `compute_key` and `generate_key`
- `qkd_engine_utils.c`: this file contains the engine code that is common to the client and the server. It includes the implementation of the function to bind the engine to the OpenSSL library, and the functions managing the socket communication between the ETSI emulator and the peer's (further information in Section 4.6).

In the following sections I describe in detail the functions implemented in each files.

[7] `https://wiki.openssl.org/index.php/SSL_and_TLS_Protocols`

[8] `https://www.openssl.org/docs/man1.1.0/man3/ENGINE_add.html`

[9] `https://www.openssl.org/docs/man1.1.0/man3/DH_generate_key.html`

[10] `https://www.openssl.org/docs/man1.1.0/man3/DH_compute_key.html`

[11] `https://github.com/gost-engine/engine`

4.5.1 The `qkd_engine_client.c` file

The `qkd_engine_client.c` file contains the engine code that is used by the client. In the file there are implemented three different functions:

- `client_generate_key()`: this function overload the `DH_generate_key()` function. It simply define the pair of private-public keys for the client. Since the pair of keys is not fundamental for our purposes, I decided to used two different fixed values, respectively for the private key and the public key.
- `client_compute_key()`: this function overload the `DH_compute_key()` function. This function is fundamental because is the one responsible for defining the `shared-secret`, therefore the symmetric key used for the communication between the client and the server. The function implements a server socket listening on the `2300 port` (running on `localhost`), and waiting to receive the QKD key from the QKD app of the ETSI emulator.
- `client_engine_bind()`: this function is used to bind the client engine to the library OpenSSL. It is necessary to call the overload dynamic library instead of the standard library implementation.

```
1  /**
2   * Callback registered in the client OpenSSL engine which is called
3   * when OpenSSL needs the engine to generate a Diffie-Hellman private-
4   * -key and to derive the Diffie-Hellman public key from it.
5   *
6   * Returns 1 on success, 0 on failure.
7   */
8  static int client_generate_key(DH *dh);
9
10 /**
11  * Callback registered in the client OpenSSL engine which is called
```

```
12  * when OpenSSL needs the engine to compute the Diffie-Hellman shared
13  * secret based on Diffie-Hellman parameters, the server public key,
14  * and the client's private key.
15  *
16  * Returns the size of the generated shared secret on
17  * success, -1 on failure.
18  */
19 static int client_compute_key(unsigned char *shared_secret,
20             const BIGNUM *public_key, DH *dh);
21
22 /**
23  * Bind the client engine to OpenSSL library.
24  *
25  * Returns 1 on success, 0 on failure.
26  */
27 int client_engine_bind(ENGINE *engine, const char *engine_id)
```

Listing 4.1: Function signatures of the client engine's main functions

4.5.2 The `qkd_engine_server.c` file

The `qkd_engine_server.c` file contains the engine code that is used by the server. In the file there are implemented three different functions:

- `server_generate_key()`: this function overload the `DH_generate_key()` function. It simply define the pair of private-public keys for the server. Since the pair of keys is not fundamental for our purposes, I decided to used two different fixed values, respectively for the private key and the public key.

- `server_compute_key()`: this function overload the `DH_compute_key()` function. This function is fundamental because is the one responsible for defining the `shared-secret`, therefore the symmetric key used for the

communication between the client and the server. The function implements a server socket listening on the `2333 port` (running on `localhost`), and waiting to receive the QKD key from the QKD app of the ETSI emulator.

- `server_engine_bind()`: this function is used to bind the server engine to the library OpenSSL. It is necessary to call the overload dynamic library instead of the standard library implementation.

```
/**
 * Callback registered in the server OpenSSL engine which is called
 * when OpenSSL needs the engine to generate a Diffie-Hellman private-
 * -key and to derive the Diffie-Hellman public key from it.
 *
 * Returns 1 on success, 0 on failure.
 */
static int server_generate_key(DH *dh);

/**
 * Callback registered in the server OpenSSL engine which is called
 * when OpenSSL needs the engine to compute the Diffie-Hellman shared
 * secret based on Diffie-Hellman parameters, the server public key,
 * and the client's private key.
 *
 * Returns the size of the shared secret on success,
 * -1 on failure.
 */
static int server_compute_key(unsigned char *shared_secret,
            const BIGNUM *client_public_key, DH *dh);


/**
 * Bind the server engine to OpenSSL library.
 *
```

```
26  * Returns 1 on success, 0 on failure.
27  */
28 int server_engine_bind(ENGINE *engine, const char *engine_id)
```

Listing 4.2: Function signatures of the server engine's main functions

4.5.3 The `qkd_engine_utils.c` file

The `qkd_engine_client.c` file contains the engine code that is common to the client engine and the server engine. In the file there are implemented three different functions:

- `create_socket()`: this function is used to create the server socket running on the client and the server. The socket is used to communicate with the QKD app of the ETSI emulator aiming to received the QKD key and use it as shared secret in the communication between the client and the server.
- `close_socket()`: this function is used to terminate the socket communication between the client (or the server) and the QKD app of the ETSI emulator.
- `QKD_engine_bind()`: this function is the callback of the `client_engine_bind()` and the `server_engine_bind()` functions. This function uses the engine APIs and the DH engine APIs to overload the cryptographic functions of the `DH_METHOD` structure:
  - `DH_meth_new()`
  - `DH_meth_set_generate_key()`
  - `DH_meth_set_compute_key()`
  - `ENGINE_set_id()`
  - `ENGINE_set_name()`

- `ENGINE_set_DH()`

```c
int create_socket(int port);

int close_socket(int sock);

/**
 * Bind the engine to OpenSSL library, register all the engine
 * functions.
 *
 * Returns 1 on success, 0 on failure.
 */
int QKD_engine_bind(ENGINE *engine, const char *engine_id,
                const char *engine_name, int (*generate_key)(DH *),
                int (*compute_key)(unsigned char *key,
                const BIGNUM *pub_key, DH *dh))
```

Listing 4.3: Function signatures of the util engine's main functions

4.6 QKD + OpenSSL Workflow

The previous section describes the structure of my project, and the implementation choices for the OpenSSL engine. The whole project is composed by two parts: the ETSI emulator implemented by UPM to obtain the QKD key, and my own implementation of an OpenSSL engine to use the QKD key as pre-shared key to encrypt the communication between two parties. In order to combine the two processes, I had to "abuse" the existing-based extension mechanism for Diffie-Hellman, as described in Section 4.5.

As first step to run the project is necessary to run the OpenSSL server through the bash script `start_server.sh`. If the server started successfully we get the message: `Starting server in background... OK (PID <server_pid>)`. The

file `server.out` contains debug messages, and output messages, in order to check the server engine workflow. An example is shown in Figure 4.8.

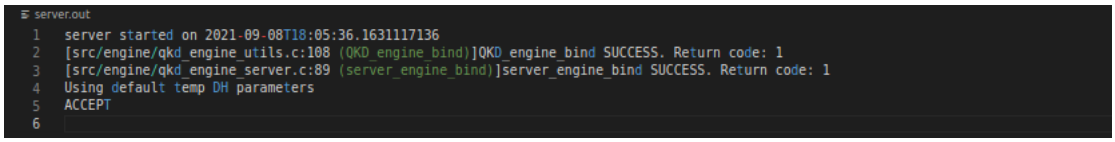

Figure 4.8: Example of `server.out` file. In the first line we can notice that the server started successfully, and the following lines show the server engine has been loaded in the OpenSSL library

After started the OpenSSL server, we can run the OpenSSL client, which will connect to the server, through the bash script `run_client.sh`. In the meanwhile, in another shell, we can run the ETSI emulator through the python scrypt `send_key.py`. The latter is responsible to run the QKD controller, the QKD nodes, and the QKD apps to retrieve the QKD key. The file `client.out` contains debug messages, and output messages, in order to check the client engine workflow. The Figure 4.9 shows the client connected to the server successfully, and the initialization of the TLS handshake. It is interesting to notice that the `client_generate_key()` function is computed after the client has received the `ServerHelloDone` message.

At this point of the TLS handshake, the `ClientKeyExchange` message should be sent by the client. The callback function `client_compute_key()` is performed before the latter step is performed, therefore, a listening socket is created, which waits to receive the QKD key from the QKD app of the ETSI emulator. In the Figure 4.10 we can see the shared secret (the QKD key) obtained by the ETSI emulator; keep in mind that this is only a prototype, the choice of a socket to make the ETSI emulator and OpenSSL communicate is not the most valuable solution according to security reasons, however, for demonstration purposes is enough to show the proof of concept.

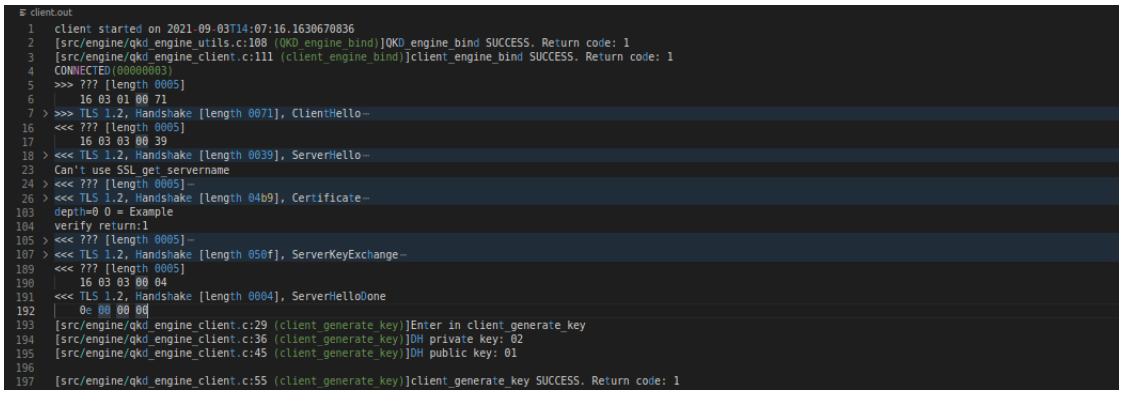

Figure 4.9: Example of `client.out` file. The first lines shows the client successfully connected to the server, and the client engine has been loaded in the OpenSSL library. Moreover the first steps of the TLS handshake are shown

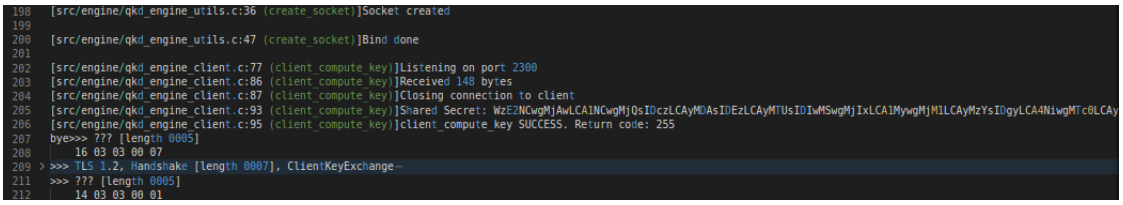

Figure 4.10: The `client_compute_key()` function has been performed: the OpenSSL client has received the QKD key from the QKD app of the ETSI emulator

Once the client has received the QKD key and the `ClientKeyExchange` message is sent, the callback function `server_compute_key()` is computed. As previously, a listening socket is created to wait the QKD key from the ETSI emulator. The Figure 4.11 shows an example of the server receiving the QKD key, and computing the `server_compute_key()` function successfully.

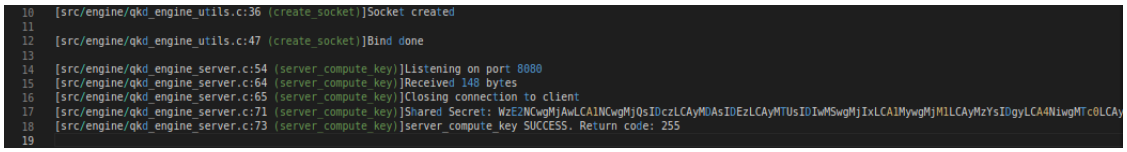

Figure 4.11: The `server_compute_key()` function has been performed: the OpenSSL server has received the QKD key from the QKD app of the ETSI emulator

Finally, the last phases of the handshake are performed. The Figure 4.12 represents a successful communication between the OpenSSL server and the OpenSSL client, using as shared secret the QKD key obtained from the ETSI emulator.

Notice that TLS 1.2 was used for this simulation, further information on this implementation choice is given in the next section.

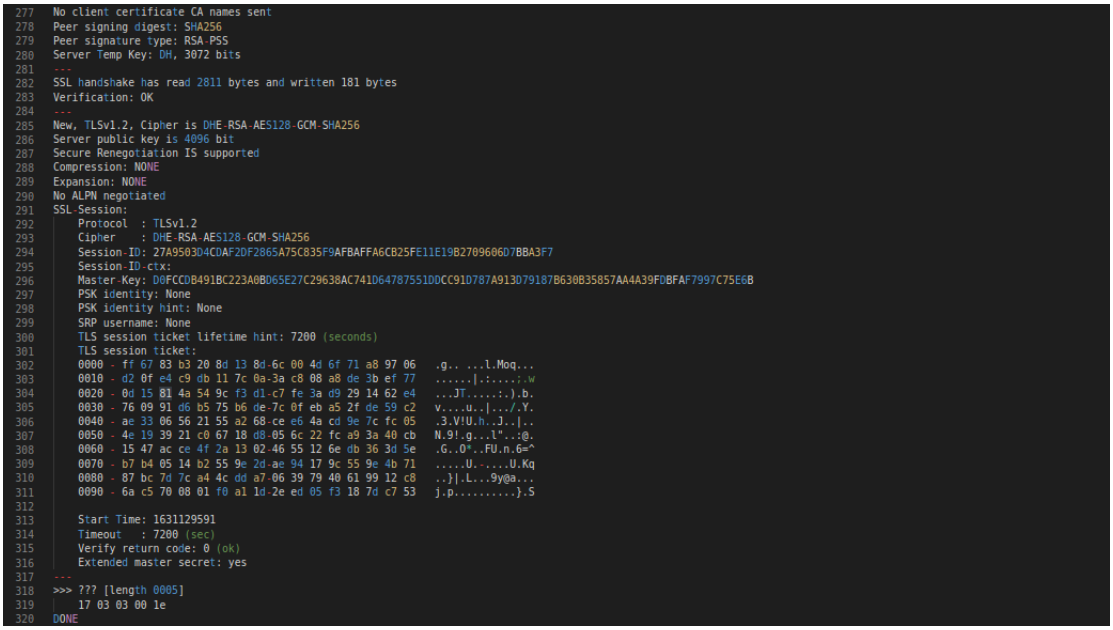

Figure 4.12: Final phases of the TLS handshake, terminated successfully. Both the client and the server has received the QKD key.

4.7 Encountered Challenges and Limitations

Implementing an engine to introduce QKD in OpenSSL was really challenging for several reasons. The first reason is related to the complexity of the library. OpenSSL library is really huge, and hard to understand, in particular, it is not so obvious to comprehend how to introduce third parties in order to extend it. The GitHub repository of the Hackathon organized by RIPE labs helped me a lot to get the first track, however, I found the implementation process very difficult because I had to

go through the documentation of the library to understand which callback functions should I have used, and how to code an engine to extend the library. It took me almost 2 months to fully understand how to develop the process, and how to design the project.

Another challenge was related to the combination of the ETSI emulator, implemented by Universidad Politécnica de Madrid, and the OpenSSL engine that I created. The first one is written in Python code, instead for the latter, I had the constraint to implement it in C code since it is the main language of the whole OpenSSL library. I have never had experience in making two different programming languages communicate, and the most plausible solution for me was to implement a socket on which the QKD key is passed. This kind of solution cannot be considered an optimal solution and it has security issues, however, my goal was not to implement a go-to-market project. A demonstration of the actual possibility of introducing the QKD technology in a widely used library such as OpenSSL, was my target.

The last aspect that needs to be considered is the compatibility of the project that I implemented. The current solution works successfully with TLS 1.2, and OpenSSL 1.1.1, which are both stable versions. However, OpenSSL is currently working on a new version of the library, and most of the methods that I used, even the engine mechanism, are considered deprecated in the new implementation. The version has not yet obtained a stable state, and the majority of applications are still based on version 1.1.1. However, an upgrade of my project with the new version of the OpenSSL library, and with TLS 1.3 should be taken into consideration in the future.

5 Conclusion

This thesis describes a demonstrative approach to integrate QKD technology into the OpenSSL library. The first part of the thesis presents the general problem to the reader: the quantum threat. In particular, Chapter 2 presents the current state of cryptography, defining asymmetric cryptography as one of the main cryptographic techniques used by communications and systems. Subsequently, I explain quantum cryptography and the reasons why it is considered a risk to classical cryptography. Chapter 2 concludes by presenting two possible technologies that might be able to address quantum attacks: Post-Quantum Cryptography and Quantum Key Distribution.

Chapter 3 focuses on the Quantum Key Distribution protocol which is introduced by Section 2.4. Chapter 3 aims to give general information about the protocol in order to comprehend the entire context and the technologies involved in my internship activity. Despite the benefits of Quantum Key Distribution, which seems to offer unhackable systems and communications, Chapter 3 concludes by presenting some security issues of Quantum Key Distribution. The security issues presented make unhackable communications still out of reach.

The first part of the thesis is composed by Chapter 2 and Chapter 3. The second part of the thesis consists of Chapter 4, which describes my activity internship at Cefriel. Chapter 4 begins by describing the challenge to be solved and the goals of my internship. Then, I present my solution to integrate QKD technology into the

OpenSSL library. Firstly, I describe all the technologies involved in my project in order to fully understand the context, then I present the different implementation choices, explaining the reasons made to develop the project. A full description of the project's workflow is offered as a practical demonstration of my solution. The final result is not a go-to-market product, however, it aims to demonstrate the feasibility of integrating QKD technology into a widely used library such as OpenSSL.

Finally, I conclude Chapter 4 by describing the main challenges that I encountered during the implementation of the project, and the limitations of the final result. The limitations presented give ideas for possible future works.

5.1 Future works

Section 4.7 concludes Chapter 4 by describing the limitations of my work to integrate QKD technology into OpenSSL. These limitations from another perspective can be seen as ideas for possible future works.

This project is composed by two different processes: (i) the ETSI emulator, implemented with Python language, (ii) and my OpenSSL engine, implemented with C language. The usage of different programming languages made it hard combining the two technologies. In order to make the two processes communicate, I implemented a socket communication, which is used to send the QKD key retrieved from the emulator. This implementation choice lacks security, however, this project is just a prototype to demonstrate the feasibility of integrating QKD technology into TLS communications. Moreover, this limitation is linked to the challenge presented in Section 3.4: the lack of an adequate security interface between the end users/applications and the quantum nodes. Thus, finding a solution to transmit the key from the ETSI emulator to the OpenSSL engine, means finding a solution for one of the biggest issues in Quantum Key Distribution networks. Section 2.4.1 mentions Post-Quantum Cryptography: this technology

can be considered to transmit the key from the quantum nodes to the end users/applications.

Finally, the solution described here is compatible with TLS 1.2 version, and OpenSSL 1.1.1 version. Currently, the OpenSSL developers' community is implementing version 3 of the library, which supports TLS 1.3 version. TLS 1.2 and OpenSSL 1.1.1 are both stable versions and are used in many services and communications. The newest version of OpenSSL is not declared stable yet, therefore, I decided to rely on the most recent stable version of the library. However, a migration to the latest library version should be considered once OpenSSL version 3 will be declared stable.

5.2 Future Collaborations

On September 27th, I presented the results obtained in this work to my supervisors Enrico Frumento, and Francesco Morano from Cefriel. Moreover, at the final presentation partecipated also Paolo Comi, Fabrizio Bianchi and Maurizio Barbaro from Italtel.

Italtel has implemented a Software-Defined Networking (SDN) QKD framework, which abstracts the internal architecture of a QKD system providing a common standardized middleware. The SDN QKD framework provides a Local Key Management System and standard interfaces to allow communications between QKD systems and end applications.

At the end of the presentation, Italtel showed their interest in the project I presented, and they intend to continue this project by integrating my solution with the SDN QKD framework, implemented by them, at the beginning of the new year.

I am really satisfied with the results achieved, I knew it was a hard work that required a lot of effort, but receiving a lot of interest for a project that I implemented individually paid all the fatigue.

References

[1] M. E. Hellman, "An overview of public key cryptography", *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, 2002.

[2] S. Wolf, "Unconditional security in cryptography", in *Lectures on Data Security: Modern Cryptology in Theory and Practice*, Springer, 1999, pp. 217–250.

[3] D. E. Denning, "Is quantum computing a cybersecurity threat? although quantum computers currently don't have enough processing power to break encryption keys, future versions might", *American Scientist*, vol. 107, no. 2, pp. 83–86, 2019.

[4] S. Wiesner, "Conjugate coding", *SIGACT News*, vol. 15, no. 1, pp. 78–88, 1983.

[5] B. Gilles, "Brief history of quantum cryptography: A personal perspective", *IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security, 2005.*, pp. 19–23, 2005.

[6] B. Gilles, "Relativized cryptography", *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pp. 383–391, 1979.

[7] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental quantum cryptography", *Journal of Cryptology*, vol. 5, no. 1, pp. 3–28, 1992.

[8] C. H. Bennett, G. Brassard, and A. K. Ekert, "Quantum cryptography", *Scientific American*, vol. 267, no. 4, pp. 50–57, 1992.

[9] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing", *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, pp. 175–179, 1984.

[10] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring", *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, 1994.

[11] D. Mermin, "Breaking rsa encryption with a quantum computer: Shor's factoring algorithm", *Lecture notes on Quantum computation*, pp. 481–681, 2006.

[12] J. Preskill, "Quantum computing in the nisq era and beyond", *Quantum*, vol. 2, p. 79, 2018.

[13] E. Gibney, "The quantum gold rush", *Nature*, vol. 574, no. 7776, pp. 22–24, 2019.

[14] N. A. R. C. Frank Tavares. (Oct. 23, 2019). "Google and nasa achieve quantum supremacy", [Online]. Available: `https://www.nasa.gov/feature/ames/quantum-supremacy`.

[15] L. Gyongyosi and S. Imre, "A survey on quantum computing technology", *Computer Science Review*, vol. 31, pp. 51–71, 2019.

[16] M. J. Nene and G. Upadhyay, "Shor's algorithm for quantum factoring", in *Advanced Computing and Communication Technologies*, Springer, 2016, pp. 325–331.

[17] V. Bhatia and K. Ramkumar, "An efficient quantum computing technique for cracking rsa using shor's algorithm", *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, pp. 89–94, 2020.

[18] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying grover's algorithm to aes: Quantum resource estimates", in *Post-Quantum Cryptography*, Springer, 2016, pp. 29–43.

[19] Enrico Frumento, Nadia Fabrizio, Paolo Maria Comi. (Jan. 25, 2021). "Il progetto quantum-secure net: Sviluppo di un prodotto europeo di quantum key distribution", [Online]. Available: `https://www.difesaonline.it/evidenza/cyber/il-progetto-quantum-secure-net-parte-13-la-minaccia-quantum-alla-crittografia-moderna`.

[20] D. D. Maria Korolov, "What is quantum cryptography? it's no silver bullet, but could improve security", *CSO Insiders*, 2019. [Online]. Available: `https://www.csoonline.com/article/3235970/what-is-quantum-%20cryptography-it-s-no-silver-bullet-but-could-improve-security.html`.

[21] G. Mone. (Jul. 1, 2020). "The quantum threat", [Online]. Available: `https://cacm.acm.org/magazines/2020/7/245691-the-quantum-threat/fulltext`.

[22] É. Kelly. (). "Germany to invest €2b in quantum technologies", [Online]. Available: `2021-05-11`.

[23] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, *et al.*, "Status report on the second round of the nist post-quantum cryptography standardization process", *US Department of Commerce, NIST*, 2020.

[24] T. Laarhoven, M. Mosca, and J. Van De Pol, "Solving the shortest vector problem in lattices faster using quantum search", in *International Workshop on Post-Quantum Cryptography*, Springer, 2013, pp. 83–101.

[25] O. Regev, "Quantum computation and lattice problems", *SIAM Journal on Computing*, vol. 33, no. 3, pp. 738–760, 2004.

[26] Research Institute. (Apr. 16, 2019). "A guide to post-quantum cryptography", [Online]. Available: https://medium.com/hackernoon/a-guide-to-post-quantum-cryptography-d785a70ea04b.

[27] C. E. Shannon, "Communication theory of secrecy systems", *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[28] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[29] D. Mayers, "Unconditional security in quantum cryptography", *Journal of the ACM (JACM)*, vol. 48, no. 3, pp. 351–406, 2001.

[30] P. W. Shor and J. Preskill, "Simple proof of security of the bb84 quantum key distribution protocol", *Physical review letters*, vol. 85, no. 2, p. 441, 2000.

[31] R. Alléaume, C. Branciard, J. Bouda, T. Debuisschert, M. Dianati, N. Gisin, M. Godfrey, P. Grangier, T. Länger, N. Lütkenhaus, *et al.*, "Using quantum key distribution for cryptographic purposes: A survey", *Theoretical Computer Science*, vol. 560, pp. 62–81, 2014.

[32] Tsai, Chia-Wei and Yang, Chun-Wei and Lin, Jason and Chang, Yao-Chung and Chang, Ruay-Shiung, "Quantum key distribution networks: Challenges and future research issues in security", *Applied Sciences*, vol. 11, no. 9, p. 3767, 2021.

[33] C. Elliott and H. Yeh, "DARPA quantum network testbed", BBN Technologies Cambridge: New York, Tech. Rep., 2007. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA471450.pdf.

[34] M. Peev, C. Pacher, R. Alléaume, C. Barreiro, J. Bouda, W. Boxleitner, T. Debuisschert, E. Diamanti, M. Dianati, J. Dynes, *et al.*, "The SECOQC quantum key distribution network in Vienna", *New Journal of Physics*, vol. 11, no. 7, p. 075 001, 2009.

[35] Maria Korolov, Doug Drinkwater. (Mar. 12, 2019). "What is quantum cryptography? it's no silver bullet, but could improve security", [Online]. Available: https://www.csoonline.com/article/3235970/what-is-quantum-cryptography-it-s-no-silver-bullet-but-could-improve-security.html.

[36] ETSI, "Quantum Key Distribution (QKD); Application Interface", ETSI, Tech. Rep., 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/004/02.01.01_60/gs_qkd004v020101p.pdf.