Universitat Rovira i Virgili (URV) y Universitat Oberta de Catalunya (UOC)

Master in Computational and Mathematical Engineering

FINAL MASTER PROJECT

Area: Cryptography

Towards Post-Quantum OCSP

Author: Maurici Toribio

Advisors: Dr. Oriol Farràs, Francesc Orozco

Barcelona, June 14, 2022

Dr./Dra. (name) Oriol Farràs Ventura, certifies that the student (name) Maurici Toribio has elaborated the work under his/her direction and he/she authorizes the presentation of this memory for its evaluation.

University advisor's signature:

FARRAS VENTURA ORIOL - 79271456R Digitally signed by FARRAS VENTURA ORIOL - 79271456R Date: 2022.06.15 14:58:05 +02'00'

Mr./Mrs. (name) Francesc Orozco, certifies that the intern (name) Maurici Toribio has elaborated the work under his/her direction and he/she authorizes the presentation of this memory for its evaluation.

Company advisor's signature:

FINAL PROJECT SHEET

Title:	Towards Post-Quantum OCSP
Author:	Maurici Toribio
Advisors:	Oriol Farrás, Francesc Orozco
Date (mm/yyyy):	06/2022
Program:	Master in Computational and Mathematical Engineering
Area:	Cryptography
Language:	English
Key words	OCSP, Post-Quantum, PKI

"A little bit of math can accomplish what all the guns and barbed wire can't: a little bit of math can keep a secret."

Edward Snowden

Acknowledgments

I would like to thank my advisor, Dr. Oriol Farràs for his help and for guiding me during all the thesis. I also would like to thank all the PKI team, and specially Francesc Orozco for helping me in everything they could.

Finally, I would like to thank the UOC and URV institutions as well as Entrust Datacard for providing me with all the needed resources to make this thesis possible.

Abstract

Since the early stage of internet, the necessity of establishing secure connections between two parties has been a fundamental part of it and a continuous topic of study. Public Key Infrastructures or PKI's have been a key part of its development by means of the use of public key cryptography. Shor presented an algorithm in 1994 that was a turn around in the field because it makes possible to break the most currently used algorithms in PKI's, as it has the capability to factor integer numbers in polynomial time using quantum computers.

In the past years, and due to the fast improvements in the quantum computers, the topic has become more relevant, as the possibility that in the next decade traditional algorithms become obsolete has become more realistic. Therefore, NIST is in process of standardization of several algorithms that could replace current ones, that are considered to be vulnerable. But for the transition to post quantum, the necessity to study these new algorithms in real protocols is needed. In this work, one of the NIST final round digital signature schemes will be implemented and tested in a PKI protocol. The case of study will be the OCSP protocol, which is a widely used protocol in PKI's to check whether a certificate is revoked or not. The Post-Quantum algorithm to be studied will be Crystals-Dilithium. Various benchmarks are done comparing the performance of this PQ algorithm with two currently widely used, RSA and ECDSA. Results show 50% better CPU performance than RSA and 40% less than ECDSA, in the other hand, it requires around 10x more bandwidth.

Keywords: PKI, Cryptography, Crystals-Dilithium, OCSP, NIST, Post-Quantum

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Since the beginning of the Internet, cryptography has been used to establish secure connections between parties across the globe. The TLS protocol is a good example because enables browsing the internet using HTTPS securely, among many other applications. Public Key Infrastructures (PKI), by means of public key cryptography, are also a key part of the development of Internet, and they provide many secure-related services. For example, PKI's make use certificates and Digital Signatures (DS), a public-key schema, to provide authenticity, in turn, DS are based mainly in the hardness assumption of the RSA and Discrete Logarithm (DL) problems.

In recent years, research in quantum computers has increased. The possibility of the apparition of quantum computers for commercial use in the next decades has become realistic. Quantum computers, by means of the Shor's algorithm are able to factor integer numbers and compute discrete logarithms in polynomial time. Currently, many public cryptosystems are built on these assumptions, so the existence of a powerful enough quantum computer would put in risk the security of all digital communications, including transactions, card payments and blockchain based systems.

The National Institute of Standards and Technology NIST, which is responsible for the standardization of some of the most used algorithms nowadays such as RSA ECDSA, or AES, recently started a competition to find new quantum resistant public-key cryptographic algorithms. Currently, in the third round of the competition, there are proposals for several digital signature algorithms as well as Key Encapsulation Mechanisms. It could take some time until the threat of Quantum Computers is real, but the standardization of these new algorithms is still a long time process. Additionally, the implementation of these new algorithms and their effectiveness in current protocols and in real case scenarios like in PKI's has to be still proven.

1.2 Objectives of the project

The main objective of this project is to test one of the post-quantum algorithms that are in process of standardization in a real case scenario. First, ensure that the implementation can be possible. Then, perform some benchmarks to compare the performance with currently used algorithms. Finally, study the result differences in terms of CPU usage, memory, bandwidth and latency and discuss the differences encountered.

The case of study will be the OSCP protocol, an important protocol in the PKI system. The main function of this protocol is to provide real time information of the revoked status of a certificate. Whenever a request is made about the revocation status of a certificate, the OCSP server checks that it has this information, then signs the response with his private key and sends it back to the user. The Post-Quantun algorithm picked to be implemented is one of the three digital signature finalists, CRYSTALS-Dilithium, an algorithm based in lattice cryptography. The current algorithms to be compared with are RSA and ECDSA.

1.3 Confidentiality

The project will make use of a commercial product, called Entrust Validation Authority owned by Entrust Datacard, and adapt it for its study. This product implements the OCSP protocol and is suitable for this study. For this reason confidentiality is needed. This is possible thanks to the collaboration between the University Rovira i Virgili and the company Entrust Datacard. For the case of the code for the digital signature CRYSTALS-Dilithium, the post-quantum algorithm, it has open source libraries in various languages and is free to be used.

There will be two versions of the thesis. The first one will contain the complete work and no information will be omitted and must be kept confidential. The second one will be a reduced version of the first one, and may be released for public view.

1.4 Organization

This document is divided in two main parts. The first part it is more conceptual and introduces several important ideas of cryptography and Public Key Infrastructure. In this part we explain some of the most important cryptosystems. The second part is more practical, where the case of study is implemented and studied. In this part, the case of study is explained in detail, as well as what modifications need to be done and what will be measured. Finally, we explain the setup of the benchmarks, show all the results obtained and discuss the results.

The first part is divided in three chapters. Chapter 2 explains the main concepts of cryp-

tosystems, with special focus in digital signatures. Also, it explains the main digital signature schemes used. Chapter 3 introduces the PKI architecture and explains their main components. Additionally, the OCSP protocol is described in detail. Chapter 4 describes the Post Quantum current situation and shows the main PQ crypto schemes. Then it explains the lattice based cryptography and Crystals Dilithium digital signature scheme.

The second part is divided in two chapters. Chapter 5 describes the Entrust Validation Authority and its main components and architecture as well as its Post-Quantum adaptation for the project. Finally, in Chapter 6 we perform different benchmarks comparing the original implemented algorithms RSA and ECDSA with Dilithium. The results obtained are shown together with the conclusions.

Chapter 2

Current Cryptography Standards

In this chapter, the basics of cryptography are explained. First, we introduce the concept of secret key encryption, explaining the differences between symmetric and asymmetric cryptography. Then, we explain RSA, one of the most important public key algorithms. After that, we explain Digital Signatures, which represent a basic part in the PKI's. Finally, we present some of the most important Digital Signature schemes: DSA, RSA and Elliptic Curves.

2.1 Symmetric vs Asymmetric Cryptography

Traditionally, confidentiality of data is provided by secret key cryptosystems or cryptographic algorithms. We can distinguish basically to kind of cryptosystems, ones that use Symmetric Cryptography and one that use Asymmetric. These systems are also called private-key cryptography and public-key cryptography respectively. For the case of symmetric cryptography, an example of these algorithms is the *Advanced Encryption Standard* (AES), which was selected as an official *Federal Information Processing standard* (FIPS) in 2002 for the United States. AES is currently used overall in telecommunications [6]. In a symmetric secret key cryptosystem the communication partners, which we call Alice and Bob, agree on a secret key before they secretly communicate. For this key agreement they can use a secure channel, or they can use the Diffie-Hellman key exchange protocol which does not require a secure channel.

Figure 2.1 shows the process of a secret key encryption in a private-key cryptosystem. If Alice wants to send a confidential message to Bob she uses the secret key to encrypt the data to be kept confidential, producing as a result a ciphertext. Alice sends the ciphertext to Bob, and upon receiving it, Bob uses the secret key that he has exchanged with Alice to decrypt the ciphertext. This kind of cryptosystems are also referred as symmetric cryptosystems since Alice and Bob have the same key.

With the development of an open computer network such as the Internet, with its billions

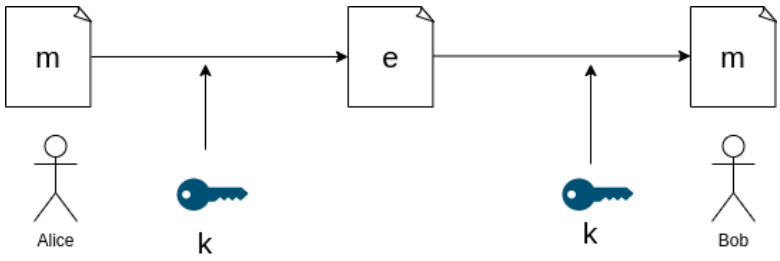


Figure 2.1: Symmetric Cryptography.

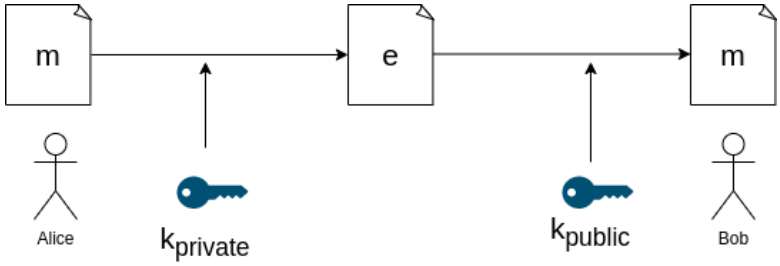


Figure 2.2: Asymmetric Cryptography.

of users, exchanging secret keys can be impractical. One option would be to use a centralized center such as the mobile standard GSM, which uses many interconnected key centers. Each GSM provider operates in one or more centers, however key distribution by a centralized key center has the disadvantage that can access to all the secret messages [6].

Another option would be to use a public-key cryptosystem. The main idea behind it is that two different but related keys are used, one for encryption and one for decryption. Decryption works with the decryption key only. As the decryption key cannot be determined from the encryption key, the encryption key can be public. In order to Bob receive confidential messages, he uses a key pair consisting of an encryption key and his corresponding decryption key. Bob keeps the decryption key secret while his encryption key is made public. The encryption key is called Bob's public key, the corresponding decryption key is called Bobs private key. Once the keys are generated and Bobs public key is published Bob can receive confidential messages from anyone and no further distribution is required. In public key cryptosystems is only necessary to make the keys accessible. Public-key cryptosystems are also called asymmetric cryptosystems, in Figure 2.2 a public key cryptosystem scheme is shown.

Public key encryption no only provides confidentiality, but it can be used to implement protocols. The first and most frequently used public key cryptosystem is the RSA algorithm, which is named after its inventors Rivest, Shamir and Adleman. The main idea upon this algorithm is based is the assumption that factorization of integer numbers is a hard problem, given that the prime numbers are sufficiently large. It is also based in the concept that discrete logarithm problem is hard to solve efficiently, which means in polynomial time $\mathcal{O}^k(n)$. The

basic explanation of the RSA is algorithm is the following [6]:

First, to generate his secret key and the corresponding public key, Bob selects two large numbers p and q and computes their product.

$$n = pq$$

Bob also chooses an integer e with the following conditions:

$$1 < e < \phi(n) = (p-1)(q-1)$$
$$gcd(e, (p-1)(q-1)) = 1$$

Then Bob computes an integer d with:

$$1 < d < (p-1)(q-1)$$
$$d\,e \equiv 1 \mod (p-1)(q-1)$$

This number d can be computed by means of the *Extended Euclidean Algorithm*. Finally, Bob's public key is the pair (n, e) and his corresponding private key is d, as we can see the private key is derived from the public key. The number n is called the RSA modulus, e is also called the encryption exponent and d also called the decryption exponent. If we want to encrypt a message, if m is the plaintext message to be sent, we can obtain the encrypted message c, doing the following:

$$c = m^e \mod n$$

And to decrypt the message the inverse operation is realized:

$$e = m^d \mod n$$

The private key can be computed from the public key e if p and q are known. Therefore, is the adversary is able to find the prime factorization of n, then she can easily find Bobs private key. The RSA schema as presented is not secure as it is vulnerable to some attacks. Nevertheless, RSA is the basis for other algorithms like RSA-OAEP or ElGamal as well as for some digital signature schemas, that will be explained later.

In practice public-key encryption is rarely used to encrypt large amounts of data, as encryption/decryption times of public key algorithms vary from the symmetric key ones in various orders of magnitude. Furthermore, algorithms like RSA, encrypts "messages" of limited size, as defined by PKCS#1. Therefore, usually a Hybrid Encryption is preferred, where the symmetric

key is encrypted and distributed using a public cryptosystem, and then used to encrypt the data [6].

2.2 Digital Signatures

Another use of cryptography is the possibility to provide integrity or data authenticity, for example, of software. A *Message Authentication Code* (MAC), which is a Symmetric Cryptosystem, can be used to provide integrity. But in many contexts it is not possible to use them, for example when we need to provide authentication. In these cases an asymmetric authentication mechanism is required, also called Digital Signatures (DS). In a digital signature scheme, the signing key is also referred as Bob's private key which is used to calculate the digital signature of the document. People that wants to verify the signature uses the verification key, which is also called public key. This works because of the property that, in the public-key systems, the private key is infeasible to be calculated from the corresponding public keys. An example of a signature scheme is shown in the Figure 2.3.

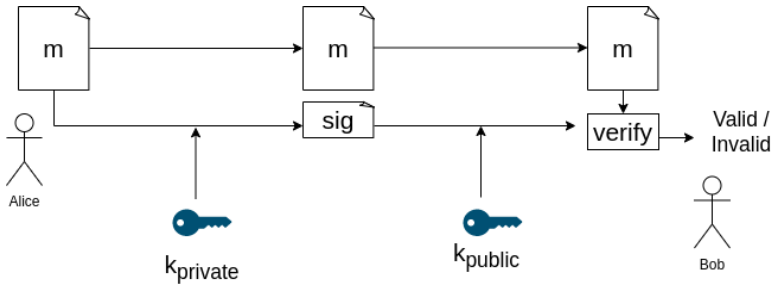


Figure 2.3: Digital Signatures.

The Digital signatures can be used not only to prove integrity and authenticity of data, also they prove non-repudiaton [6]. Because of those numerous applications, digital signatures are extremely important cryptographic tool. Suppose that Bob signs data such as an email of a bank transaction digitally. By verifying this signature, Alice convinces herself that the data origin is Bob (something that not provide a MAC) and that the data has not been altered.

2.3 RSA Digital Signature

The most widely used signature algorithm is the based in RSA scheme described before. The key generation for this algorithm works exactly as in the RSA cryptosystem presented before. To describe the operations of signing and verifying the previous notation is used. The private key d becomes the signature key, and the public key (n, e) it is the public key or verification

key. In addition to signature and verification keys, a publicly known collision-resistant hash functions is used

$$H : \{0,1\}^* \to \{0,1....,n-1\}. \tag{2.1}$$

Then the signature **s** of a string of bytes $m \in \{0,1\}^n$ is:

$$s = H(m)^d \mod n$$

To verify the signature, the verifier uses the signers public key (n, e), determines the hash value $H(D)$ and checks that:

$$H(m) = s^e \mod n.$$

If holds, then the signature is valid. Otherwise, is invalid. The schema explained before is called RSA-FDH, and it uses the approach of signing the hash of the message instead of signing the whole message, a typical approach called hash-and-sign paradigm. The function H plays an important role, and practical attacks are found in case that the output of H is too small, for this reason RSA-FDH is not practically deployable, and some variations are used as RSA-PKCS# 1 v1.5, which introduces an additional step [11].

2.4 ECDSA signatures

ECDSA stands for *Elliptic Curve Digital signature algorithm* and is also a widely used signature schema together with the RSA. It belongs to the family of the *Elliptic Curve Cryptography* (ECC), and it provides the same level of security as RSA with considerable shorter keys and signatures. ECDSA is based on the DSA algorithm, and both algorithms rely on the hardness assumption of the *Discrete Logarithm* (DL) problem. In DSA we have a private key d as a random integer between $0 < d < q$. The public key is calculated as follows [11]:

$$e = g^d \mod p.$$

Where g, p and q are fixed. The constant p is a prime number $2^{L-1} < p < 2^L$ where $521 < L < 1024$, and q is a prime divisor of $p-1$ with length of 160 bits. Then, a single use private key k and public key s is needed. First k calculated choosing a random integer with $0 < k < q$, this ephemeral public key is then:

$$r = (g^k \mod p) \mod q.$$

Then, if we have a message m, by means of the ephemeral private key generated in the previous step, as well as the main private key, it is possible to calculate the signature as

follows:

$$s = (k^{-1}(H(M) + xr)) \mod q,$$

where H is a Hash function, and finally the signature is (r, s). To verify the signature the user calculates the following values and verifies that $v = r$.

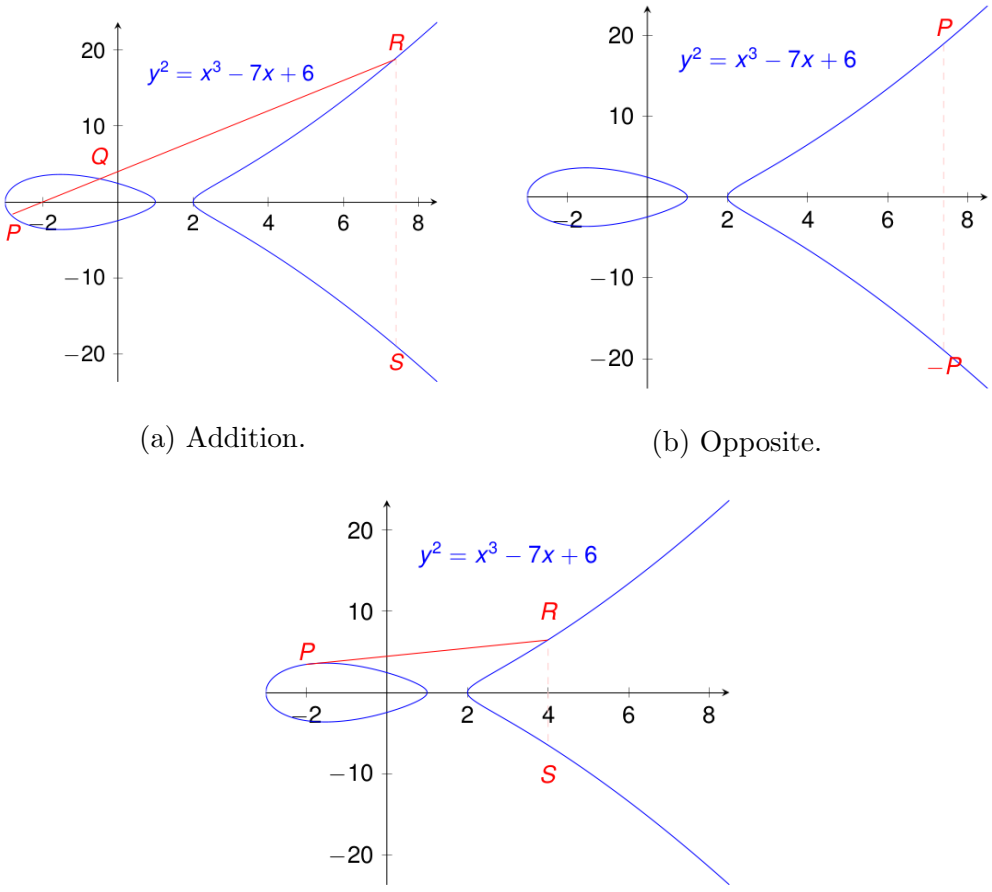
$$\begin{aligned} w &= s^{-1} \mod q \\ u_1 &= w \cdot H(M) \mod q \\ u_2 &= r \cdot w \mod q \\ v &= (g^{u_1} e^{u_2} \mod p) \mod q \end{aligned}$$

Contrary to the DSA, in ECDSA the calculations are made over elliptic curves, which is a special form of polynomial equation of the form $y^3 = x^3 + a \cdot x + b \mod p$. An elliptic curve over the cyclic group Z_p is defined as the set of all pairs $(x, y) \in Z_p$ which fulfill the previous equation, together with an imaginary point of infinity $\mathcal{O}$ [18]. Also, another condition is that the curve must be non-singular, which geometrically speaking means that is has no intersections.

As we are operating in a finite field, for cryptography purposes, we are interested in plotting the curve over a prime field Z_p. For this reason, a set of group rules for the cyclic group must in which the previous equations operations are performed is defined. Given two points and their coordinates $P = (x_1, x_2)$ and $Q = (x_2, y_2)$ the following operations are explained as geometric constructions, however simple coordinate geometry can be applied to express them through analytical expressions o formulas [18]. The operations defined are:

- **Point Addition**. To compute $R = P+Q$, draw a line between P and Q and to obtain the point of intersection between the elliptic curve and the line, then mirror the intersection to obtain R.
- **Point Doubling**. In this case to obtain $R = 2P$, draw a tangent line through P and obtain the second point of intersection and mirror it to obtain P.
- **Identity** is defined as $P + O = P$, where O is located at plus infinity along the y-axis.
- **Inverse**. To find $-P$ it can be found mirroring the point into the x-axis and find it in the curve.

In the Figure 2.4 there is an example of the *addition*, *doubling*, and *opposite* operations in the elliptic curve $y^2 = x^3 - 7x + 6$.



(a) Addition.

(b) Opposite.

(c) Doubling.

Figure 2.4: Operations over Elliptic Curves extracted from [11].

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer Factorization	RSA	1024	3072	7680	15360
Discrete logarithm	DH, SA, Elgamal	1024	3072	7680	15360
Elliptic curves	ECDSH, ECDSA	160	256	384	521
Symmetric-key	AES, 3DES	80	128	192	256

Table 2.1: Bit length for different security levels extracted from [18].

A distinguished feature in public-key algorithms is that they require long operands and keys. The longer the operands and keys the more secure the algorithms become. In order to compare the security of different algorithms usually the term *security level* is used. What this means is that, an algorithm is said to have a "security level of n bit" if the best know attack requires 2^n steps. In the case of asymmetric algorithms, for a level of security of n the key has a length n, but for the case asymmetric algorithms the relationship is not straightforward [18]. The Table 2.1 shows a comparative between the key size and security level of different symmetric and asymmetric cryptosystems [18].

Chapter 3

Public Key Infrastructure

In the previous chapter, we introduced the basic concepts of cryptography, as well as some of its most important cryptosystems. In this chapter we explain the *Public Key Infrastructure* or PKI, that are part of these public-key cryptosystems. First, we introduce the concept of the PKI, why it is needed and what it provides in terms of security in communication. After that we explain what are the certificates, which is an important cryptographic tool in PKI system. These certificates are generated by means of digital signatures described before. Then, we describe some of the main components of a PKI. Those are the *Certificate Authority* (CA) and *Registration Authorities* (RA), which are responsible for the issuing of certificates, and finally we describe the *Certificate Revocation Lists* (CRL) and the *Validation Authority* (VA), which are part of the revocation lifecycle of a certificate.

3.1 Introduction to PKI

The main idea behind public-key cryptography is that two strangers should be able to communicate securely, for example if Alice wants to send a message to Bob, he can do it by means of the Bob's public key, thanks to that, Alice is able to encrypt the message and send it to Bob securely. But how are the keys generated, distributed and trusted? Here is where PKI's come into play. *Public Key Infrastructures* (PKI's) enable the use of public key cryptography in open networks and on the Internet, ensuring the distribution of keys during all its lifecycle. In order to ensure a secure communication between two or more parties a PKI is generally considered to be associated with this primary services [6]:

- **Authentication** is the assurance to one entity that another entity is who it claims to be.
- **Integrity** is the assurance to an entity that data has not been modified (intentionally or

unintentionally) from the original, for example, to prevent malicious modifications in a software.

- **Confidentiality** is the assurance to an entity that no one can read a particular piece of data without authorization.
- **Non-Repudiation** is the assurance that prevents an entity from denying a particular action, for example a money transfer or sending an email.

In order to provide this services, the PKI ensures that the key management is set up correctly during all its whole lifecycle. In a key lifecycle, the following main phases can be defined [14]: the phase of *key generation*, where key pair is created. The *key establishment* phase, that makes sure that the keys are reachable. The *key usage* phase where the private key is used to decrypt or sign data, also the users can encrypt data or verify signatures, this can be done by using the public key of the others. The *key storage* phase, which ensures the correct safekeeping of the keys, as well as providing backups and preventing loss. Finally, there is the *key destruction* phase.

In the first phase, the keys must be generated securely. This can be done letting the users generate their own key pair. However, due to technical or security reasons sometimes it is not suitable. For example because the computer can be exposed to internet and can be infected by malware. A better option could be the use of dedicated hardware or smart cards. But usually the use of specific hardware as *Hardware Security Modules* (HSM) is recommended as smart cards are limited in computational resources. Another important question is how the keys are generated. RSA keys are generated using two random prime numbers, and each of those prime numbers is generated by means of a *Pseudo Random Number Generator* (PRNG) [6]. Thus ensuring that this PRNG are generated correctly is extremely important. Otherwise, it could lead to vulnerabilities such as the *Return of Coppersmith Attack* (ROCA). Exploiting this attack affected in the past thousands of electronic identity cards in Estonia and in other counties as well [2].

In the *key establishment* phase, the most important task, is to make the public keys available to the users. Those who use public keys must be able to verify authenticity and validity. If the authenticity of Bob's public key cannot be guaranteed the adversary can replace the Bob's public key with his own public key, which then Oscar may be able to decrypt messages that were encrypted by Bob or sign documents in the Bobs name. This is known as a *Man in the Middle Attack* (MITMA).

The last task of a PKI, which belong to the *key destruction* phase, is to deal with the problem whenever a private key becomes insecure, for example if the smart card where Alice stores her private key is stolen the corresponding public key must no longer to be used to

verify signatures of Alice. PKI users must be informed in this case. Also, if the public key cryptosystem is broken, for example with a quantum computer, or some vulnerability is found, such as ROCA, then all public key that have been issued must be invalidated.

3.2 X509 Certificates

One of the most important tasks of a PKI is to provide authenticity proofs of the public keys. The most important proof are the certificates. To explain the concept of a certificate suppose that Alice wishes to verify the digital signature issued by Bob. Alice needs to convince herself of the authenticity of that public key. Certificates can be described as data structures that bind public keys to entities and that are signed by a third party [6]. If Alice trusts the third party that signed the certificate and also trusts the signature verification key of the third party, then verifying the signature of the certificate convinces Alice of the authenticity of the public key of Bob. So the certificates reduce the trust in a public key of an entity to the trust in some authority.

The most important certificate standard is the X.509 standard, which are specified on a *Request for Comments* (RFC), concretely in *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* [5]. RFC's are memorandums, published by the *Internet Engineering Task Force* (IETF) and they are used to describe methods, research and innovations that are applicable in the working of Internet and other connected systems. The structure of the certificates is specified using the *Abstract Syntax Notation Version 1* (ASN.1) as a specification language, which allows describing complex data structures. ASN.1 permits encoding rules, in the case of the X509 certificates, they are encoded according to the *Distinguished Encoding Rules* (DER). Below there is an overview of what constitutes a Certificate in ASN.1 encoding:

Listing 3.1: ASN.1

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }
```

As can be seen from the RFC5280 [5] the first element is the *tbsCertificate* which is of type *TBSCertifictate* (another data structure defined in ASN.1), this part contains all the necessary information to describe an entity, and it also contains its public key. All this information is then signed using a digital signature schema.

The second element is the *signatureAlgorithm*, which describes the algorithm used, and is

of type *AlgorithmIdentifier* or OID. An OID is basically a number to identify cryptographic algorithms which are standardized as well as other objects, this algorithm must appear in the *tbsCertificate* and outside. Finally, the third element which is the result of the signature of the *tbsCertificate*. This digital signature is appended at the end as a bit string. The contents of the signed document part or the *TBSCertificate* are described below. All elements except the *issuerUniqueID*, *subjectUniqueID* and *extensions* fields are mandatory:

- **Version** indicates the version of the certificate, currently there are three versions, and is identified with an integer.
- **Serial Number** is the unique identifier for this certificate relative to the certificate issuer which corresponds to a 20 bytes integer. The combination of issuer name and serial number makes the certificate unique.
- **Signature** indicates the algorithm identifier (that is, the Object Identifier, or OID, plus any associated parameters) of the algorithm used to calculate the digital signature on the certificate. For example, the OID for SHA-1 with RSA might be present, indicating that the digital signature is an SHA-1 hash encrypted using RSA.
- **Issuer** is the *Distinguished Name* (DN) of the CA that issued the certificate and must always be present.
- **Validity** indicates the window of time or validity period that this certificate should be considered valid. This field is composed of two dates, the *Not Valid Before* and the *Not Valid After*. This dates/times that may be represented in UTC Time or in Generalized Time.
- **Subject** indicates the DN or distinguished name of the certificate owner and must be non-null unless it is *Subject Alternative Name* (SAN) form is used int the extensions section.
- **Subject Public Key Info** is the public key identified with a sequence containing an OID, associated with the subject. This field is mandatory in this section as well as outside the *tbsCertificate* part.
- **Subject Unique ID**. Used when the same DN is used by different entities, the use of this field is not recommended because it makes the certificate more complicated.

In practice the contents of the X.509v1 Certificate and v2 were insufficient, for this reason later on the version 3 of x509 standard appeared. This newer version allows content extensions and are widely supported by current PKI's. Some of the most used extensions are listed below:

- **Authority Key Indentifier** or AKI is the Hash of the issuer's public key, calculated with SHA-1 algorithm. The information of the issuer field may not be sufficient to identify the public key because the issuer may have multiple keys, then this extension allows applications to identify the public key of the issuer, which must be used to verify the signature.

- **Subject key identifier** is the hash value of the key which corresponds to the certificate. It is used by applications to compare the public key in the certificate with other public keys, and it is useful if the owner of the certificate has several public keys. As well as the AKI, this is calculated by means of SHA1.

- **Key Usage** indicates what the public key of the certificate can be used for, for example, for *Digital Dignatures* or *Non-Repudiation*, among others.

- **Subject Alternative name** (SAN) it is used to bind the public key to additional names such as IP addresses, *Domain Names* (DNS), or *Uniform Resource Identifiers* (URI's).

- **Issuer Alternative Name** analogous to the SAN, used to bind the Issuer to some alternative name.

- **Extended Key Usage** which indicate some additional usages to the key usage field, for example for *Time Stamping* or *Revocation Status Signing*.

- **Further extensions** additional extensions, for example *certificatePolicies* and Poly *Mappings*, *BasicConstraints* etc. Additionally, some other extensions called private extensions may be defined for certain context.

3.3 Certification Authority

A Certification Authority (CA) is one of the core components of a PKI. A CA is responsible for binding the corresponding subject name to a public key. In the concept of a PKI, this act of binding is called certification, and as discussed before this action occurs in the form of a data structure that is referred as a certificate [1]. This act of certification, often also called issuing, is basically the signing using digital signatures the certificate with the private key of the CA, often called issuer of the certificate. Because the certificate is digitally signed, the information contained in it is protected from an integrity perspective. If the public key of the issuer's certificate is trusted, so it is the data contained in it. A CA can take different representations, depending on the trust model [1], for example in an enterprise domain, the company would be

responsible for issuing the certificate to its employees and thy would place their trust in the company's CA certificate.

In order to obtain a certificate, a *Certificate Signing Request* (CSR) must be firstly generated, which usually contains the public key with some additional information which identifies the applicant as described in the PKCS#10 specification. Then this CSR is sent to the corresponding CA which issues the certificate. If the CSR does not contain the public key, the CA takes care of the key generation on behalf of the owner, storing the private key in a secure environment and providing the user with a PIN, similar to what it is used in smart card. A CA is also responsible for issuing other PKI's certificate entities as CRL's or Validation Authorities, or even other CA. Also, is the responsibility of a CA to revoke the certificate in case of requested. Finally, there is also the possibility that a certificate is self-signed, and that the issuer of the certificate is the same as the subject, meaning that the signature corresponds to the public key contained in the certificate.

3.4 Registration Authority

During the *certification phase*, there is an important process that involves registration of the user or entity, called *registration phase*. This is done by a component in a PKI called *Registration Authority* or RA. This process it is mainly based in establishing and confirming the identity of the certificate owner [6]. Registration usually requires a lot of interaction between certificate owner and the responsible for this registration function, sometimes requires a physical presence. For example, during the process of certification of an electronic identity document or a passport. This also initiates the certification process with the CA. This process generates all the necessary information, keys, CSR, revocation information and distributing all the information among the PKI components.

Although a CA could be responsible for this registration process, usually it is better to off-load all this functionality from directly implementing it this component, and giving this responsibility to a RA. But it should be noted that regardless of the functions, a RA is never allowed to issue certificates, CRL, or give revocation status of certificates [1].

3.5 Hierarchical Trust

In public-key cryptography, the trust of is based in the idea of trusting the public key of a given authority. To establish this trust there are different methodologies and models, in this section we introduce the *Direct Trust Model* and its extension the *Hierarchical Trust Model*, which are widely used in PKI's.

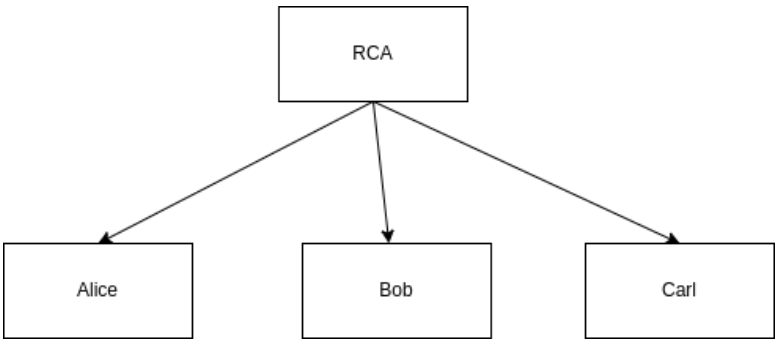


Figure 3.1: Simple hierarchical PKI.

The most fundamental trust model is the *Direct Trust Model*. In this model, one can trust in a public key authenticity if the key is directly obtained by its owner or if him confirms in a convincing way the authenticity the key to the user. For this reason, how this key is obtained and trusted is a fundamental security issue because if a third party obtains and modifies the key during the process all the process falls apart. In order to obtain the key, many of the applications use preinstalled public keys. They can be preinstalled in the browser, in the email or even in the operating system, for example [6].

There are a lot of trust models which depend on the *Direct Trust Model* such as *Hierarchical Trust Model* or the *Web Trust Model*. Among all of them *Hierarchical Trust Model* is widely used in PKI's and is based in the X.509 standard. In a hierarchical model based PKI, public keys are certified by *Certification Authorities*. The CA assumes liability for the authenticity of public keys that they certify [6]. A hierarchical model it is typically represented as an inverted tree with a CA on top which is called the root CA. This CA acts as the root of trust, also called the *trust anchor* as all the user certificates rely on it. The intermediate nodes are called subordinate CA and the leaves are the end users [1].

The Figure 3.1 shows a very simple hierarchical PKI. In this PKI, a single CA, which is also the root, has issued certificates to the end entities Alice, Bob and Carl. Previously, This CA must have issued a self-signed certificate to itself and this certificate contains the public key that is to be used to verify the signatures of Alice, Bob and Carl's certificates. In the case of this self-signed certificate, both subject and issuer are the CA itself. In this case all the entities in the PKI establish direct trust in the trust anchor, Since the PKI users trust the trust anchor they also thrust the authenticity of the public key of Alice, Bob, and Carl [6].

Usually, and due to scalability issues, is more common that a CA issues certificates to other subordinate CA's to delegate the job to issue certificate to end users or even to other subordinate CA's.

In this case of a more complex structure with subordinate CA's, authenticity of a public is established via a certification path or chain. An example of a more complex PKI can be seen

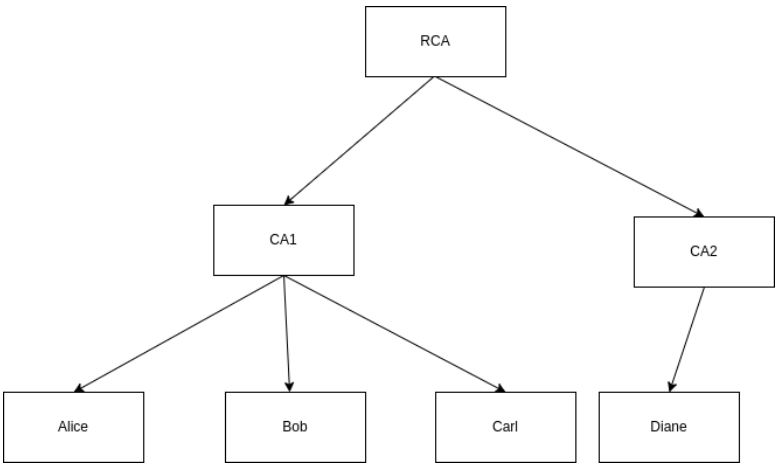


Figure 3.2: Complex hierarchical PKI.

in the Figure 3.2. In this PKI, the root CA RCA, has issued two subordinate CA certificates to CA1 and CA2. In turn, this CA's have issued certificates to end users. If someone for example wants to trust the public key of Diana, it must use the certificate chain which consists of three certificates C^{RCA}_{RCA},C^{RCA}_{CA2}, C^{CA2}_{Diana}, being the upper index the public key and the subindex the signature. The chain starts with the self-signed certificate of the root CA, in the next certificate the root CA certifies the public key of CA2. In the third certificate, CA2 certifies the public key of Diana [6].

3.6 Certificate Revocation

The validity of a certificate is usually of various years, and although the current cryptosystems are secure, it can happen that the certificate must be invalidated before this period expires. This could be due to multiple reasons, for example, to a private key that has been compromised or a job status.

In terms of the key lifecycle in a PKI, this would belong to the *key deletion* phase. During this phase the PKI's CA is informed that the certificate is revocated, together with the related reason. Then, there must be some way of informing to the involved entities about this certificate revocation.

Certificate revocations can be implemented in different ways, One method is to use periodic publication mechanisms which can be implemented in numerous ways. The most used of these publications methods is the so called as *Certificate Revocation Lists* (CRL's). In the other hand, there are also online query mechanisms such as the *Online Cerificate Status Protocol* (OCSP) [1].

3.6.1 Certificate Revocation Lists

One of the PKI components responsible for this functionality are the *Certificate Revocation Lists* or CRL. A CRL is basically a signed data structures that contain a list of revoked certificates. The digital signature appended to the CRL provides the integrity and authenticity of this CRL [1]. A CRL structure is defined in the RFC5280 [5] together with X509 Certificates. It uses similarly ASN.1 for its encoding.

As a CRL is basically long list with revocation information, they can be requested once to the corresponding responsible. It also can be kept locally and cached in order to be read without having access to Internet, which could be really useful in some cases. On the other hand, and as a disadvantage, the publication of a CRL is done periodically. Thus, it could be some delay between the actual revocation status, and what the CRL contains.

The signature of CRL can be done by the same issuer CA that issues the certificates. But usually this process is designated to another entity in order to simplify the functionality of the components in a PKI. Following the hierarchical trust model described, the CA would issue a CRL entity which would take care of this functionality.

Through the years, different versions of CRL standards where released, basically due to vulnerability concerns of the initial versions. Currently, version 2 standard is mostly used. This version solved all the previous problems by introducing the notion of extensions much the same as X.509 certificates. A list of the fields contained in the *TBSCertList* (To be Signed) CRL is seen below [5]:

- **Version** is optional and contains the version of the encoded CRL. It can be either 1 or 2, and if not present version 2 will be used. In case of extensions filed is used, this field must specify v2.
- **Signature** contains the OID of the digital signature algorithm used to calculate the signature of the CRL as well as specifying the hash algorithm, for example RSA2048 with SHA256.
- **Issuer** similar to the certificates, it contains the *Distinguished Name* (DN) of the CRL issuer, responsible for singing it, and must always be present. Additional alternative names may appear in the *issuerAltName* extension.
- **This Update** contains the issuing time of the CRL, which must be encoded as either UTCTime or Generalized Time.
- **Next Update** indicates the time when the next CRL will be issued, and current CRL will no longer be valid. This field it is designated as optional in X.509, and must be encoded as UTC time or generalized Time.

- **Revoked Certificates** contains the list of revoked certificates referenced by their serial number. Additionally, the date on which the revocation occurred must be specified. Additional extension may be necessary sometimes with additional information of each certificate.

- **Extensions** field allows more information to be added with each individual revocation. Similar to the certificates extensions, *Authority Key Identifiers*, or *Issuer Alternative Names* among other information is allowed.

3.6.2 Online Certificate Status Protocol OCSP

With periodic public mechanisms it is possible to obtain the full revocation information at predictable points of time. Also, this can be stored in cache in order to be consulted later. Thus, it is possible to obtain revocation information even when the interested offline and is not connected to the CRL. Periodic public mechanism, on the other hand, have some disadvantages: files are extremely large and consulting them is really time-consuming. Additionally, storing them may need a lot of space which is not available for example in mobile devices. Furthermore, the validity period and publications of CRL is in the order of days and the information contained may be not up-to-date [6].

In order to solve these problems, an online Query mechanisms can be used alternatively. These mechanisms return revocation status at running time using some predefined protocol. Among these types of protocols the most used is the *Online Certificate Status Protocol* (OCSP) which is defined by IETF as RFC2560 [20]. This protocol uses basically HTTP *GET* and *POST* methods with some additional headers as defined in the RFC [20]. OCSP protocol allows clients to query an OCSP server about the revocation status of individual certificates. The main advantage is that it returns more up-to-date information than a CRL. OCSP server responsibility are sometimes delegated from a CA to a so-called *Validation Authority* (VA). In these cases, although information is fresher than CRL, the information is cached by the VA. Furthermore, some VA servers just query CRLs which eliminates this advantage. OCSP main advantage is that does not require much storage, only the revocation information must be stored. On the other hand OCSP requires applications to be online.

The OCSP protocol workflow is the following: OCSP clients send a request to an OCSP server about the revocation status of one or more certificates, optionally the request can be digitally signed. For each certificate the request submits the serial number of the certificate. The hash value of the issuers DN and the hash value of the issuers public key. This information determines the certificate uniquely. Listing below shows the ASN.1 specification of an OCSP request.

Listing 3.2: ASN.1

```
OCSPRequest     ::=   SEQUENCE {
  tbsRequest                  TBSRequest,
  optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

TBSRequest      ::=   SEQUENCE {
  version             [0]     EXPLICIT Version DEFAULT v1,
  requestorName       [1]     EXPLICIT GeneralName OPTIONAL,
  requestList                 SEQUENCE OF Request,
  requestExtensions   [2]     EXPLICIT Extensions OPTIONAL }
```

Next, The OCSP server responds to the request with a message digitally signed. If the status of more than one certificate is requested, then the answers contains information about the status of each requested certificate. The revocation answers can be the following: *Good* which indicates that the certificate is valid. *Revoked* indicating that the certificate has been revoked for some reason. And *Unknown* which indicates that the certificate is not known by the OCSP server and that is unable to give any answers about its status. Note that receiving a positive answer does not mean that the certificate is still valid as it could be expired. Finally, if a protocol error has occurred the OCSP sever answers with a message error, and in this case the response is not signed. Below a list of the fields as defined in [20] containing an OCSP response is shown:

- **OCSP Response Status** indicates the status of the OCSP response, It can be: *Successfull*, *malformedRequest*, *internalError*, *tryLater*, *signRequired* or *unauthorized*.
- **Response Type** contents the type of the OCSP response as a OID.
- **Version** indicates the version of the OCSP protocol used.
- **Responder ID** is the key hash of the OCSP responder that is used to identify the Validation Authority that signed the response.
- **Produced At** is the time when the OCSP responder signed the response as UTC Time.
- **Certificate ID** containing various fields: the issuer name hash, Issuer key hash and Serial number of the certificate as well as the hash algorithm used to calculate these hashes.
- **Certificate Status** contains the status of the certificate that can be *revoked*, *good* or *unknown*.

- **This Update** contains the most recent time at which the status being indicated is known to be correct by the responder.
- **Next Update** this field contains the time when the response is no longer valid, usually the valid duration time of a OCSP response are 8h.
- **Signature Algorithm** contains the algorithm identifier by means of which the response is signed by the server.

OCSP servers may be operated by various authorities. For example, it could be the same certificate issuer or could be a delegated authority such a VA. Using the same CA could result in a security risk because the same key used to sign the responses, which usually is exposed to the internet. For this reason usually a dedicated OCSP service is established. This server has a distinct DN which is different form the certificates issuers DN. If this is the case, the OCSP server must have a certificate issuer with the extended key usage extension set to the value *OCSPSigning*.

Chapter 4

Post Quantum Cryptography

In this chapter we introduce the term *Post Quantum* (PQ) Cryptography, explaining why Shor's algorithm and quantum computers could have a big impact in the current cryptosystems, and in PKI's as well. Then we explain the measures that ha been taken to counteract the effects of a PQ world. Also, introducing the new PQ algorithms that are in process of standardization. Finally, there is an introduction of Lattices, which is one of the main field of study in PQ. There we introduce Crystals-Dilithium, which is a *Digital Signature* schema that is based in lattice cryptography.

4.1 Shor's algorithm

Cryptographic schemes security rely upon mathematical problems that are assumed hard to solve. In particular the public key cryptosystems basically one way functions. These basically rely on the RSA and discrete logarithm problems DL. In RSA, the public key is a product $N = pq$ of two secret prime numbers p and q and, so the security of RSA relies critically on the difficulty of finding the factors p, q and N. The best known factorization of a number of n bits is of the order of $e^{2n^{1/3}(\log n)^{2/3}}$ and until now, all algorithms that could break RSA or DL require an exponential number of steps [12].

However, in 1994, Shor introduced a fast quantum algorithm to find the prime factorization of any positive integer N [21]. This quantum algorithm would require a number of steps of the order of $n^2((\log n)(\log\log n))$, meaning that the number of steps required is quadratic to the number of bits n. Shor also introduced a similar algorithm able to solve the discrete logarithm problem in polynomial time. This enables breaking *Elliptic Curve Cryptography* (ECC), a popular alternative to RSA, relying on DL [12].

These algorithms, when applied to public-key sizes for RSA and ECC, require billions of operations on thousands of logical qubits (bits of quantum computers). Scaling to such number

Name	Function	Pre-quantum security level	Post-quantum security level
Symmetric cryptography			
AES-256	encryption	256	128(Grover)
GMAC	MAC	128	128(no impact)
SHA-256	hash function	256	128(Grover)
Asymmetric cryptography			
RSA-3072	signature	128	broken (Shor)
256-bit ECSDA	signature	128	broken (Shor)
256 bit ECDH	key exchange	128	broken (Shor)

Table 4.1: Security Level of traditional algorithms against *Post Quantum* attacks extracted from [3]

of qubits is difficult, and currently infeasible. Although at some point it could encounter fundamental obstacles, but no such obstacles have been identified. Thus, quantum computers with such power are likely to be possible at some point in the future [3].

More cryptographic systems are affected by an algorithm that Grover introduced in 1996. This algorithm is the foundation for most, but not all possible applications that have been identified for quantum computing. Grover originally described his algorithm as searching an unordered database of size N using $\sqrt{N}$ quantum queries [3]. Applying and developing this concept into symmetric cryptosystems and hash functions, it decreases the execution time of the attacks. But the decrease is not enough to make it vulnerable as these are no polynomial-time attacks.

Grover's algorithm speedup is not as dramatic as Shor's algorithm. If qubits operations are small enough and fast enough, then Grove's algorithm will threaten many cryptographic systems that aim 2^{128} security, but it can be solved easily doubling the size. In the table 4.1 there is a summary of the impact that the Shor and Grove algorithms have in current cryptosystems [3].

4.2 NIST Roadmap

Due to the concern that quantum computers could completely break cryptosystems, many researchers have begun to investigate in the *Post-Quantum Cryptography* (PQC) field, also called quantum resistant or quantum-safe cryptography. The goal of this field is to develop cryptographic algorithms that would be secure against both quantum and classical computers. There have been through the years a lot of proposals, including lattice-based cryptosystems, code-based cryptosystems, multivariate cryptosystems, hash-based signatures, and others. However, for most of these proposals, further research is needed in order to gain more confidence in their

security.

In 2016, the *National Institute of Standards and Technology* (NIST) decided that should be prudent to begin to develop standards for PQ. Previous cases have shown that the transition will not be as simple as replace an algorithm by another. As a significant effort will be required to develop, standardize and deploy post quantum cryptosystems, a call for proposals started.

In 2017, the first round of the competition started, in this round 49 different *Key Encapsulation Mechanisms* (KEM), which are encryption algorithms, and 22 *Digital Signature* schemes where presented. In 2019, there was the second round of the NIST, in this round only 17 KEM's and 9 Signature algorithms where left from the previous round. Then, in June 2020 the 3 Round begun. At that point only 4 KEM algorithms for encryption and 3 Signature schemes are left, with also 8 alternate candidates. Currently the competition is still in the 3rd round. It is expected that in 2023 there will be the release draft standards for the three finalists [16].

The cryptosystems that progressed to the third round of the NIST can be classified in the following families [8]:

- **Code-based schemes**. The security is based in the difficulty of decoding vectors to find the shortest error vector. This kind of scheme is more typical in KEM than in digital signatures.
- **Isogeny-based schemes**. This security depends on the difficulty of recovering a secret isogeny between a pair of elliptic curves, more used also in KEM's.
- **Lattice-based scheme**. The security is based in the difficulty of finding vectors in a lattice that are relatively short or relatively close to some target vector.
- **Multivariate schemes**. The scheme depends on the difficulty of solving systems of quadratic or higher degree multivariate polynomials.
- **Symmetric schemes**. They depend on the security of symmetric cryptographic primitives such as hash functions or block ciphers.

Another way to categorize the digital signature schemes is based in the framework to construct these primitives [8]:

- **Hash and sign**. These schemes are constructed from trapdoor one-way functions.
- **Hash-based**. These schemes follow the Work of Lamport and Merkle to construct a signature from a hash functions.
- **Fiat-Shamir** These schemes are constructed by using Fiat-Shamir transform, together with a post-quantum Identification scheme(IDs)

	Lattice-based	Multivariate-based	Symmetric-based
Hash-and-sign	FALCON	GeMSS Rainbow	
Hash-based			SPHINCS+
Fiat-Shamir	Dilithium		Picnic

Table 4.2: PQ Digital Signatures categorization extracted from [8].

Alg Type	PubKey(Kb)	PrivKey(Kb)	SigSize(Kb)	KeyGen	Sign	Verify
				Speed 10^6 cycles		
SPHINCS+ 192s-L3	0.04	0.1	16	206	1919	1.6
Dilithium3	1.9	4	3.2	0.25	0.43	0.18
Falcon 1.7	0.03	1.3	1.2	63	0.79	0.17

Table 4.3: PQ digital signatures comparison extracted from [8].

In the Table 4.2 a summary of the different categorization digital signature schemes can be seen [8].

NIST provided a guidance on the evaluation criteria to apply to the candidates. In this guidance there are various defined security categories in terms of the resources required to attack different algorithms. In total there are 5 categories going from 1 to 5, from less to more restrictive: Category 1 is equivalent to the resources needed for key recovery of AES-128. Category 2 is equivalent or greater than collision search or SHA3-256. Category 3 to break AES-192, category 4 is equivalent to break SHA3-384, and finally, category 5 to break AES-256. From all of them at least a category 3 variant should be presented for each algorithm. Also, NIST recommends that submissions should include parameters set that meets greater categories to demonstrate flexibility and to protect against future cryptoanalytic attacks. For example Dilithium provides parameter set to support Category 2, 3 and 5 [8]. Every one of the presented algorithms has different parameters, key sizes, and performance. In Table 4.3 the different properties for some category 3 PQ signature schemes can be seen.

The information regarding security level and speed in Table 4.3 is taken from an ETSI report with date of 2021 [8]. It has to be taken into account that the NIST round 3 is an ongoing process and this means the security of schemes may vary at some point. For example, a recent study presents several algorithmic improvements to the dual lattice attack. Dual Lattice attacks is cryptoanalysis technique used against the hard problems relying on the *Learning With Errors* (LWE) and *Learning With Rounding* (LWR) problems. These improvements considerably reduce the security level of Kyber, Saber and Dilithium [15]. Furthermore, it is possible that some new attacks appear that could break the current NIST proposed algorithms.

Recently a vulnerability has been found on Rainbow, which is one of the finalists. This attack returns the secret key after in average 53 hours of computing time on a standard laptop [4].

4.3 PKI PQ Transition

As explained, due to the concerns related Quantum Computing, NIST started a competition whose objective is finding a replacement algorithm. But whether the industry is ready to move to one of this cryptographic algorithms, specially in a Public Key infrastructure (PKI), must still be proven. Research work in this area is necessary to prepare the market for the challenges that could be encountered and help to move towards a PQ-PKI.

Specially a lot of study is needed whether or not the actual protocols are suitable to this coming new algorithms. Transport Layer Security (TLS) which is the main protocol for ensuring end-to-end encryption has had some recent advances. For example in [7], various benchmarks have been done comparing TLS with new protocols like KEMTLS, which involves the using of KEM for both confidentiality and authentication. Not only TLS but also other important protocols: such as Online Certificate Status protocol (OCSP), Enrollment Over Secure Transport (EST) protocol, or Timestamp Protocol (TPS), among other important PKI protocols, should be tested. This is because the PQ algorithm key sizes, parameters, encryption and signing times vary.

For the case of the certificates and the X.509 standard, the simplest transition to use PQ algorithms would be to put the PQ public keys and signatures directly into the existing field. This way only the standardization of new OID is needed without changes in the actual X509 standard. In the other hand, this has the disadvantage that pure PQ certificates could only be used together with PQ applications.

A better option would be if a smooth transition to PQ should be considered, where both traditional and PQ algorithms live, such certificates are called “Hybrid Certificates”. This kind of certificates can contain two or more signatures and keys in the same certificate. This way could be a good option until the PQ is completely adopted.

As the format of a dual signature is out of scope of the NIST drafts, it is up to the application to specify how to parse signatures and verify them separately.

Some IETF proposals are currently in phase of standardization in order to apply this Hybrid Certificates. One option of Hybrid Certificates, suggests placing the PQ data, such as public keys, algorithm identifiers, and signatures into non-critical x.509 v3 extensions, this method has the advantage that no big changes into the current standards are needed [22].

Another way would be to combine a collection of PQ and classical algorithms into a single composite algorithm, with individual key and signature objects, encapsulating multiple

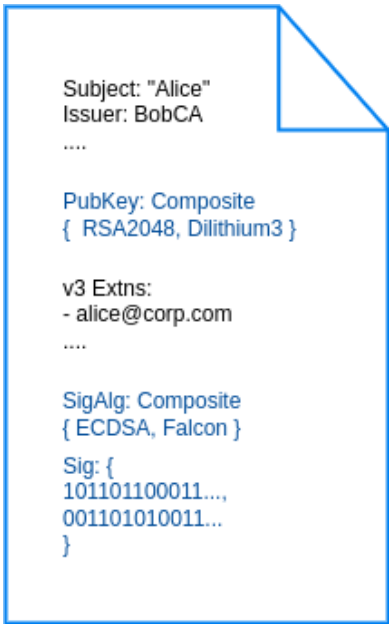


Figure 4.1: Hybrid Composite Certificate

keys and signatures. This so-called "Composite Certificates" are in an early stage of IETF Standardization. This method would require new OID's for composite structures, and has the disadvantage that backwards compatibility is tricky [17]. In Figure 4.1, an example how a composite certificate would look is shown. There the public key of Alice is a composite signature that uses RSA and Dilithium. The certificate is issued by BobCA with ECDSA and Falcon.

Another thing to take into account in a PKI is the performance of each algorithm in terms of encryption, decryption, signing, key generation etc. This is important as a PKI usually needs to scale properly with sometimes thousands or users using its components and protocols at the same time. For this reason investigations in terms of performance are being made, these studies are usually focused on the acceleration of specific hardware which could be useful in PKI's by means of the *Hardware Security Modules* (HSM). An HSM are a widely used method to store private information as keys securely. Various studies show proposals using a Software/Hardware co-design method, a technique with the goal of reaching performance target using shorter development cycle. For the case of the new PQ algorithm candidates it also allows a better approach as the pure-hardware implementation approach due its complexity [10]. Examples of this methodology are shown in the implementation and benchmarking of co-design in three Lattice-based KEM's from the NIST process [10]. Another example is in the HW/SW co-design acceleration of the Classic McEliece cryptosystem, a code-based cryptosystem from the NIST 3rd round [13].

4.4 Lattice Based Cryptography

One of the most active and promising fields in PQ cryptography is lattice based cryptography. The security of these schemes rely on the hardness of the *Shortest Vector Problem* (SVP) and its variants. These problems are related to some foundational average-case problems, called the *Short Integer Solution* (SIS) and *Learning With Errors* (LWE). NTRU, which is one of the NIST KEM candidates, is lattice based and is well studied cryptosystem. Furthermore, three of the 4 KEM's are lattice based as well as two of the three digital signature algorithms. This is because polynomial attacks are not know until now, even in quantum computers [12].

Another important point is that lattice-based cryptosystems are often algorithmically simple and parallelizable, consisting in linear operations and matrices modulo relatively small integers. For the case of lattices over certain rings, for example, the case of NTRU cryptosystem it can be specially efficient [19].

Lattices $\mathcal{L}$ are in, the simplest terms a grid of points in the space $\mathbb{R}^n$. Although lattices $\mathcal{L}$ are infinite, they always generated as an integer linear combination of some linearly independent vectors, which we call the basis $\mathbf{B} = \{b_1, ..., b_k\}$:

$$\mathcal{L}(\mathcal{B}) = \mathbf{B} \cdot \mathbb{Z}^k = \left\{ \sum_{i=1}^{n} z_i b_i, \;\; z_i \in \mathbb{Z} \right\}$$

The integer k is called the *rank* of the basis and is invariant of the lattice. From now on only full-rank lattices are used, where $k = n$ [19]. As an example, in the Figure 4.2 a 3-Dimenssional lattice is plotted, where the basis vectors of the lattice are b_1, b_2 and b_3, the linear combination of these with integers, generates a set of points in the space.

As explained, lattices, have some interesting properties, whose characteristics are that there are a set of problems involved that are hard to solve. Some PQ algorithms rely on in this conjecture and this class of optimization problems on lattices, which are intractable. One particular is the *Shortest vector problem* (SVP).

In SVP, we have a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ of some arbitrary basis $\mathcal{B}$. The problem consists on finding the shortest non-zero lattice vector $\mathbf{v} \in \mathcal{L}$ for which $||\mathbf{v}|| = \lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L}) = \min||v||$ [19]. In the Figure 4.3 a 2-Dimensional Lattice is shown as example. There the vectors b_1 and b_2 correspond to the basis vectors of the lattice, and the vector v would correspond to the shortest vector of the lattice. In this case the problems seems easy but increasing the dimensions the problem becomes really hard to solve.

The SVP is a problem hard to solve in the worst case, and it has several variants, like the *Decisional Approximate* SVP or the *Approximate Shortest Independent Vectors Problem*.

Another important hard problem in Lattices is the *Bounded Distance Decoding Problem* (BDD).

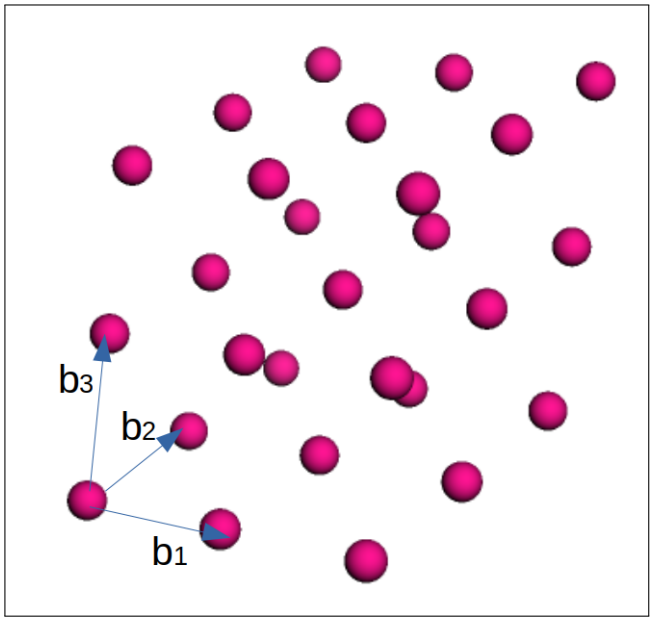


Figure 4.2: 3D Lattice

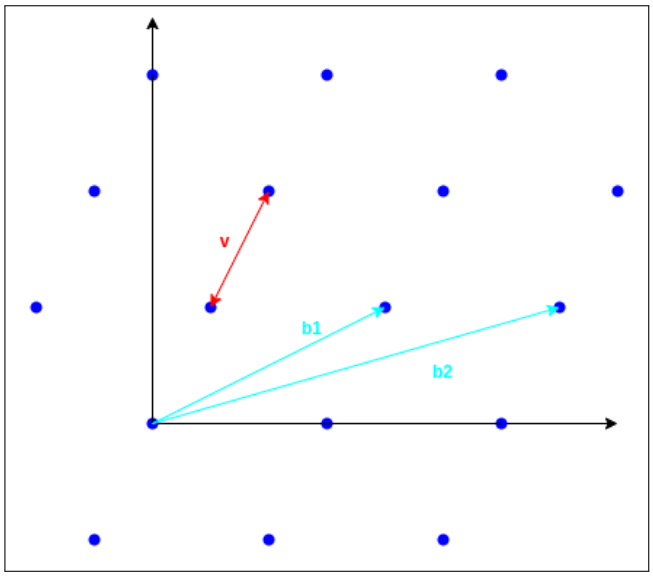


Figure 4.3: Shortest Vector Problem.

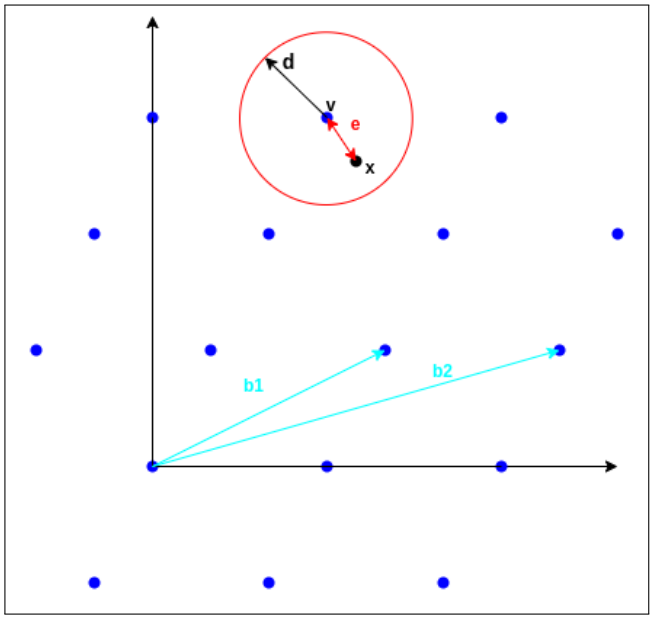


Figure 4.4: Bounded Distance Decoding Problem.

In this problem, given a lattice basis $\mathcal{B}$ of an n-dimensional lattice $\mathcal{L}$ and a target point $\mathbf{x} \in \mathbb{R}^n$ with the guarantee that the $\text{dist}(\mathbf{x}, \mathcal{L} < d = \lambda_1(\mathcal{L}/(2\gamma(n))$, where $\gamma > 1$ is an approximation factor, then, find the unique lattice vector $\mathbf{v} \in \mathcal{L}$ such that $||\mathbf{x} - \mathbf{v}|| = e < d$ [19].

The BDD problem, basically asks to find a vector that is closer to a given point $\mathbf{x}$ where the target is promised to be "rather close" to the lattice. Another related problem is the *Closest Vector Problem* (CVP) which it distinguishes from BDD because BDD asks for the uniqueness of the solution and CVP not [19]. An Example of a BDD Problem is shown in the Figure 4.4.

SVP and BDD problems are proven to be intractable in the worst-case assumption. This is not recommended in cryptography as it usually works in average-case scenarios as some problems that appear hard in the worst case turn out to be easier on the average. Nevertheless, using a first worst-case to average-case reduction for lattice problems it can proven that certain problems are hard on average as long as some related lattice problems are hard on the worst case. Also, it is possible to connect the *Short Integer Solution* (SIS) and *Learnig With Errors* (LWE) with the SVP and BDD problems respectively, where SIS and LWE are two proven hard average-case problems[19].

In a LWE problem, given as independent samples a matrix $A \in \mathbb{Z}_q^{n\,x\,m}$ and a vector $b \in \mathbb{Z}_q^m$, where n and q are positive integers. Then if we have a uniformly random vector $s \in \mathbb{Z}_q^n$ which we call secret and a small error distribution e, and the following holds:

$$\mathbf{b} = \mathbf{sA} + e \mod q$$

and the objective is to find the secrets **s**. Note that without the error term the secret would be easy to find by Gaussian elimination [9]. LWE can also be seen as an average-case *Bounded Distance Decoding* problem for some certain parameters in the lattices, where the vector **b** is relatively close to a one vector in the lattice, and the goals is to find this lattice vector or point [19].

$$\mathcal{L}(\mathcal{B}) = \left\{ \sum_{i=1}^{n} z_i b_i, \;\; z_i \in \mathbb{Z} \right\} + q\mathbb{Z}^m \tag{4.1}$$

4.5 CRYSTALS-Dilithium

CRYSTALS-Dilithium is a digital signature that is strongly secure under chosen message attacks based on the hardness of lattice problems. It is proposed by the *Cryptographic Suite for Algebraic Lattices* (CRYSTALS) and currently in the 3rd Round NIST competition.

The scheme is based int the Fiat-Shamir abort which is a framework to drive a signature scheme from a compact lattice based in an *Identification Scheme* (IDS) [9]. The security of Dilithium is based on the hardness of finding short vectors in lattices and the Learning with errors (LWE) problem. Considering the LWE Equation 4.1 we have that the matrix A together with the vector b, which consist of polynomials sampled randomly from Z_q, would be the public key. In the other hand s and e, which consist of polynomial vectors sampled from a small distribution of Z_q, would be the secret key [8].

Two operations constitute nearly the entirety of the signing and verification procedures. The first is an *Xtendable-Output Function* (XOF) which are similar to hashes but with the ability to produce an output of variable size. The XOF used in Dilithium are SHAKE-128 and SHAKE-256. The second operation consists in multiplication operations between polynomials in the ring $Z_q[x]/(x^n + 1)$. For all security levels, the scheme uses the same ring with $q = 2^{23} - 2^{13} + 1$ and $n = 256$. Varying this security simply involves doing more/fewer operations in the ring and doing more/less expansion operations in the XOF [9], this makes Dilithium flexible and easy to implement for different levels of security. Dilithium presents security levels, Dilithium-2 with a claimed security of category 2, Dilithium-3 with a security level of 3, and Dilithium-5, with a security of 5. Also, instead of SHAKE XOF mechanism Dilithium also allows the use of AES-256 in counter mode, called Dilithium-AES. This allows to make use of AES enabled acceleration hardware.

Now, the signing-verification procedure of Dilithium is explained in detail. First, the bound sizes of the coefficients are defines as follows [8]:

- **n** is the dimension of the polynomial ring R_q.

- **q** which is the modulus of the polynomial ring R_q.
- **k** and **l** are the ranks of the vector over R_q.
- $\boldsymbol{\mu}$ which is the bound on the size of the coefficients for key generation.
- $\boldsymbol{\gamma_1}$ and $\boldsymbol{\gamma_2}$ are the bound of the sizes of the coefficients for signing and $\boldsymbol{\beta}$ is the reduction in the bounds.

The parameter γ_1 is set strategically so that it is large enough that the signature does not reveal the secret, yet small enough that the signature is not easily forged [9]. Similarly to the BDD in lattices, the objective of the bounding parameters is to define the boundary distance so that the point results to be sufficiently close to a point in the lattice.

Additionally, some auxiliary primitives must be defined. Those are: SHAKE-256 as a hash function H, SHAKE-256 as a PRF, and SHAKE-128 or AES-256 in CTR mode as a XOF function. Each component of $w \in R_q^q$ can be decomposed as [8]:

$$w = w_1 \cdot 2\gamma_2 + w_0$$

Where the coefficients of $w_0 \leq |\gamma_2|$. Then, Dilithium also defines the functions $w_1 = \text{Highbits}(w, 2\gamma_2)$ and $w_0 = \text{Lowbits}(w, 2\gamma_2)$.

Dilithium keyGen

For the key generation, first two vectors $s_1 \in R_q^l$ and $s_2 \in R_q^k$ with size at most μ is generated using a PRF. Additionally, a matrix A is generated using a seed $\rho \in 0, 1^{256}$ and expanding it using an extendable output XOF function [8].

Then the public key is calculated as:

$$t = As_1 + s_2.$$

Then the public key is $(\rho, t) \in \{0,1\}^{256}$ and the private key is $(\rho,\, t,\, s_1,\, s_2) \in \{0,1\}^{256}$.

Dilithium sign

For signing, first a sample $y \in R_q^l$ is generated with coefficients between $-\gamma_1$ and γ_1. Then w_1 is calculated:

$$w_1 = \text{Highbits}(Ay, 2\gamma_2) \in R_q^k.$$

To sign, given a message M first the hash of the message with the w_1 appended is calculated using SHAKE-256:

$$c = H(M||w_1)$$

And then the signature is calculated as follows:

$$z = y + cs_1$$

A final step is needed to ensure the success of the signature:

$$||z||_{\text{inf}} \leq \gamma_1 - \beta or ||\text{LowBits}(Az - ct)||_{\text{inf}} \leq \gamma_2 - \beta$$

In case contrary the signing process must be restarted. The first check is necessary for security, while the second for both security and correctness [9].

Finally, the final signature will be the tuple (z, c).

Dilithium verify

Having the public key $pk = (\rho, t)$ in order to verify the signature $sig = (z, c)$, first the seed ρ must be expanded using XOF to obtain $A \in R_q^{k \cdot l}$. Then compute:

$$w_1' = \text{Highbits}(Az - ct, 2\gamma_2)$$

And finally check that:

$$c = H(M||w_1')$$
$$\text{and}$$
$$||z||_{\text{inf}} < \gamma_1 - \beta$$

If the previous two conditions are true, $ACCEPT$, otherwise $REJECT$.

Chapter 5

Case of Study

The case of study will be a *Validation Authority* (VA) called *Entrust Validation Authority* or EVA. The function of the Validation Authority is to provide information of the certificate revocation status using the OCSP protocol as described in the PKI Chapter 3. Another function of the VA apart from providing OCSP responses to whoever requests it, is to request the information periodically to a server with this status. This revocation information would be retrieved from a Certificate Authority and stored in a database. Finally, to sign the OCSP responses, the Validation Authority it connects to a *Hardware Security Module* (HSM) where the keys are stored for security, and sign the response before sending it. There is also the possibility to store the keys in the host and sign from there. Although this is not recommended in a production environment, we will use this method to perform the studies and benchmarks, as there is no Dilithium implementation for HSM's yet. In this chapter the architecture of this Validation Authority and its functionality are described in detail.

The details of the Entrust Validation Authority are hidden for public view due to a confidentiality agreement.

Chapter 6

Benchmarks

In this section we test the performance of the Validation Authority with different algorithms for signing the OCSP requests. The traditional algorithms that will be used for the benchmarks are RSA with keys of 2048 bits and ECDSA with keys of 256 bits, which are two commonly used security levels for these algorithms. For the case of Dilithium, Dilithium3 will be used, which achieves at least a security of 128 bits, [8] being very conservative. This security level is similar in terms of security level to the other two algorithms. For all the three cases the same benchmark-setup with the same hardware specifications is used. The benchmarks will be performed without the use of Hardware Security Module. This, is done because Dilithium is not available in an HSM yet and makes the benchmarks of the three algorithms more comparable. Using an HSM could lead to different results, either because there hardware acceleration or because the latencies between the VA and the HSM.

The performance of the Validation Authority is mainly measured in terms of the following parameters:

- How many OCSP Requests is the server capable to receive, sign and respond in a specified time, also called from now on as *Transactions per Second* or TPS. This is an important feature of a VA as it could have peaks with a huge amount of requests that should be able to respond.
- The period of time from the time that the OCSP request is sent, until that the OCSP response is received and verified. This parameter is also called Latency from now on. In this case the lower, the better, as the clients expect to receive the response as fast as possible.

Also, the Validation Authority should support handling multiple OCSP requests from different clients simultaneously, this parameter will be called concurrency. To stress the server and calculate the maximum performance of the system, different benchmarks will be made with a

different level of concurrency (multiple requests at the same time), until the maximum of the resources of the CPU is reached. Other hardware resources such as memory and bandwidth needed will also be measured, but these resources should not suppose a bottleneck.

6.1 Benchmark Setup

For the benchmarks three separate Virtual Machines (VM) are used, one will host the benchmark client, another the database and another one the Entrust Validation Authority. The software and hardware specifications are seen below:

- Processor: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
- Virtualization: VMWARE
- Cores: 8 CPUs
- Thread per core: 1
- Operating System: CentOS Linux 7 (Core)
- Linux kernel: 3.10.0-1160.53.1.el7.x86_64

A schema of the benchmark setup is seen in the Figure 6.1.

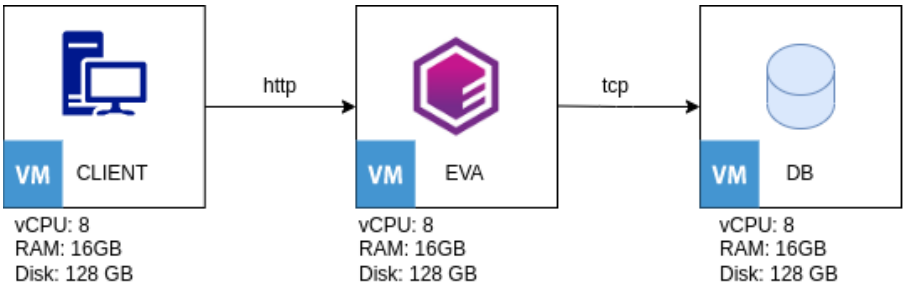


Figure 6.1: Benchmark Setup

To perform the benchmarks, first the database will be filled with all the certificates and all the revocation information. This is done in order to avoid other processes in the EVA host to consume any kind of resources, like CPU or RAM. This allows OCSP Server process to use all the resources that has available in the VM all the time. The client consists of a small command line interface program generated for the purpose of these benchmarks, and it has also the functionality to verify Dilithium Digital Signatures. To generate a OCSP request the following flags are given as arguments:

- **url** is the URL of the OCSP Responder Server.
- **n** is the number of total OCSP Requests to be sent.
- **c** is the number of concurrent requests.
- **certFile** is the certificate (PEM format) from which we want to obtain the revocation status.
- **caFile** is the CA certificate (PEM format) which issued the certificate, necessary to identify a certificate.
- **vaFile** is the VA certificate (PEM format) which will be used to verify the signature of the OCSP Response.

An example of the client usage is seen below. The command is executed in a UNIX terminal, where 100 requests are realized with a concurrency of 8 requests simultaneously to the certificate **ca.pem**.

```
$./client -c 8 -n 100 -caFile ca.pem -vaFile va.pem \\
   -certFile cert.pem -url http://127.0.0.1/eva
```

The three VM are hosted in the same data-center so very low network latencies are expected. The execution of the benchmark client and the gathering of performance results are done automatically. The relevant parameters that are measured are the following ones:

- **TPS**: Transactions (requests) per Second. Number of transactions made per second. For the benchmarks, the transaction includes also the signature verification process.
- **Latency**: Latency in *ms*, which includes: the signing, verification as well as the network latencies.
- **Server CPU**. Is CPU usage in the server VM (CPU 100% meaning all cores at 100%).
- **OCSP Memory**. Is the memory usage of the OCSP server in *MBytes*. Memory used in other processes is not included.
- **Bandwidth**. Bytes transmitted from the server to the client in *Mbps*.

All the results are calculated as an average of total number of requests made. The number of total OCSP requests performed are 100000 in all the cases, this number represents a good value to have a good average of the results. These parameters will be measured with concurrences of 1 to 1024, increasing as a power of two.

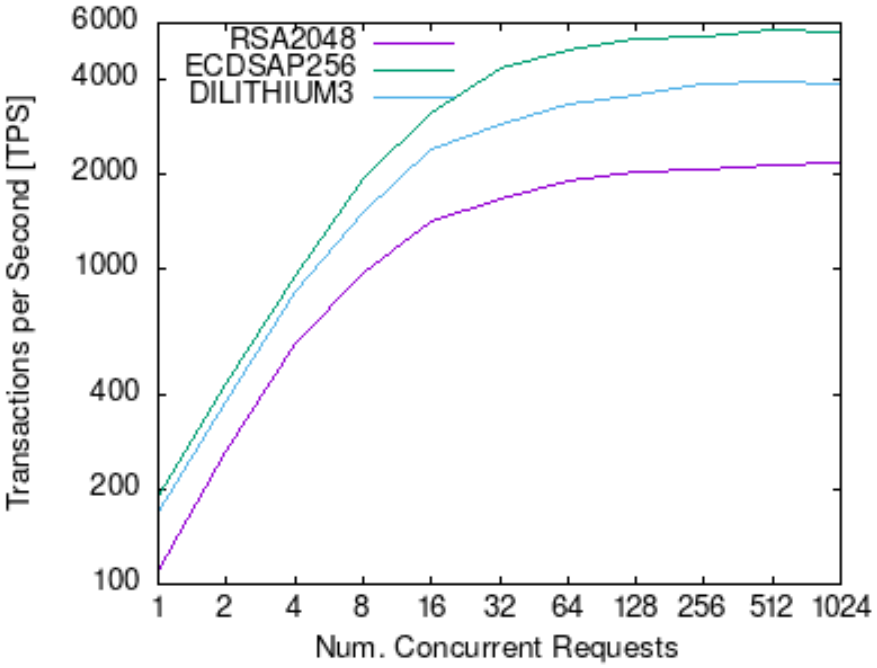


Figure 6.2: TPS for RSA, ECDSA and Dilithium

6.2 Results

In this section all measurements of the parameters described before are shown, the results are plotted in the y axis as a function of the number of concurrent requests in x where both axis are in logarithm scale. In each graph it is possible to see the results for the three algorithms RSA, ECDSA and Dilithium for comparison purposes. Figure shows 6.2 the Transactions per Second as a function of number of the concurrent requests received (in logarithmic scale) and it shows the results for the three algorithms. As it can be seen, they present different results, but the overall behavior is similar. The TPS increases linearly at the beginning and is proportional to the number of requests received, until it reaches about the 4 concurrent requests. After that, the rate starts to slow down until it stabilizes and reaches its limit at about 512 concurrent requests approximately.

Figure 6.3 shows the CPU usage of the host machine as a function of concurrency, for each case. The graph shows that as the CPU usage increases linearly at the beginning, proportional to the concurrency, until it reaches certain concurrency. After that, the CPU rate starts to slow down until it reaches about 90% of the PC usage.

Figure 6.4 shows the Latency versus concurrency, for RSA, ECDSA and Dilithium. It can be observed a flat latency below 10ms for the three algorithms, and after reaching 4 concurrent requests the latency increases proportionally at different rate for the three algorithms. In the flat region we observe a slow descent in latency from 1 to 2 and 4 concurrent requests. The

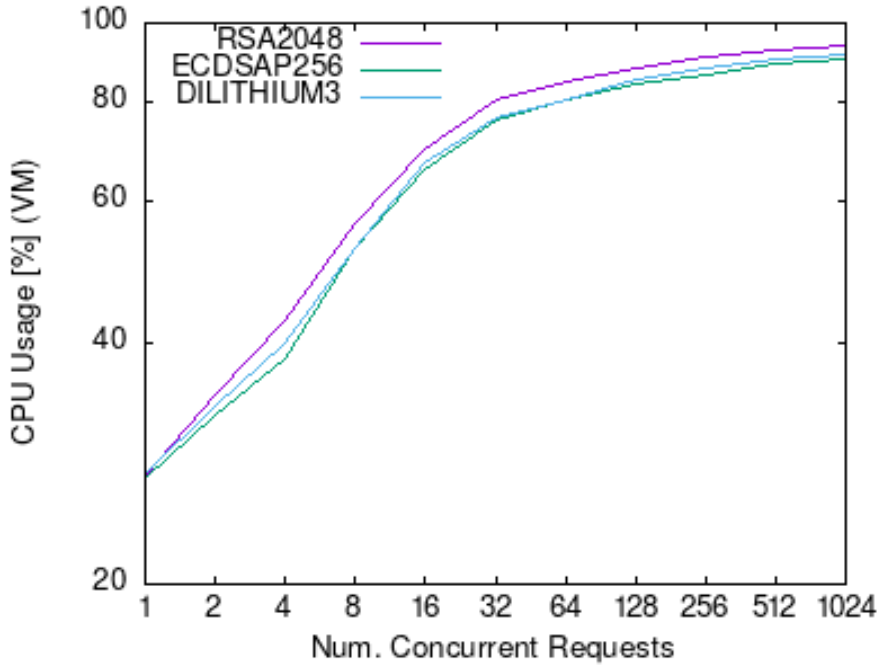


Figure 6.3: CPU Usage (%) for RSA, ECDSA and Dilithium

	RSA	ECDSA	Dilithium
Size (KB)	1.4	0.8	9

Table 6.1: Size of a OCSP Response in $KBytes$

explanation of this estrange behaviors is that the client reuses the same TCP connection, thus avoiding doing another TCP handshake and reducing a latency.

Figure 6.5 shows the bandwidth in Mbps needed by the OCSP Responder vs concurrency. Also in this case we can observe a proportional increase of the bandwidth for the three algorithms until 4 concurrent requests. After that, the stiffness starts decreasing with a logarithmic behavior until it stabilizes at some constant bandwidth. Table 6.1 shows the size in $Kbytes$ of a generated OCSP Response for each algorithm.

Finally, Figure 6.6 shows the memory usage vs concurrency used by the OCSP Sever process. In this case, the memory needed in other processes of EVA is not shown as it is not relevant for this case. An increasing exponential behavior can be observed for each algorithm.

As seen, we can distinguish three different main regions in the graphs. In the first region, TPS, CPU and Bandwidth increase linearly and proportional to the concurrency until it reaches a concurrency of 4. In this region Latency and Memory Usage are flat. After that there is a transition until TPS, CPU and Bandwidth peaks at approximately 512 of concurrency. Latency in this case increases linearly and Bandwidth exponentially.

In the Table 6.2 the maximum performance results are shown (for each algorithm). This

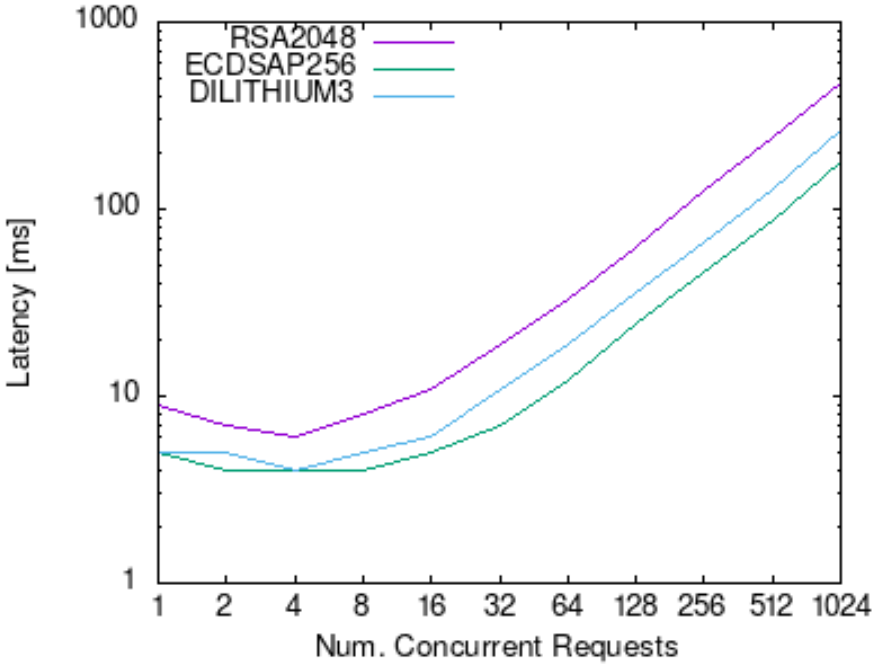


Figure 6.4: Latency (ms) for RSA, ECDSA and Dilithium

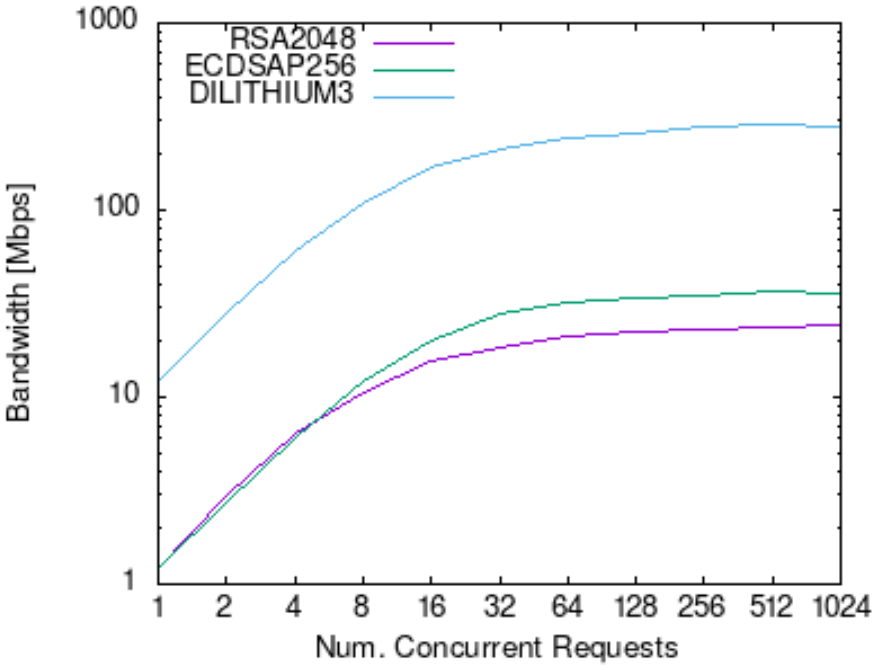


Figure 6.5: Bandwidth Usage (Mbps) for RSA, ECDSA and Dilithium

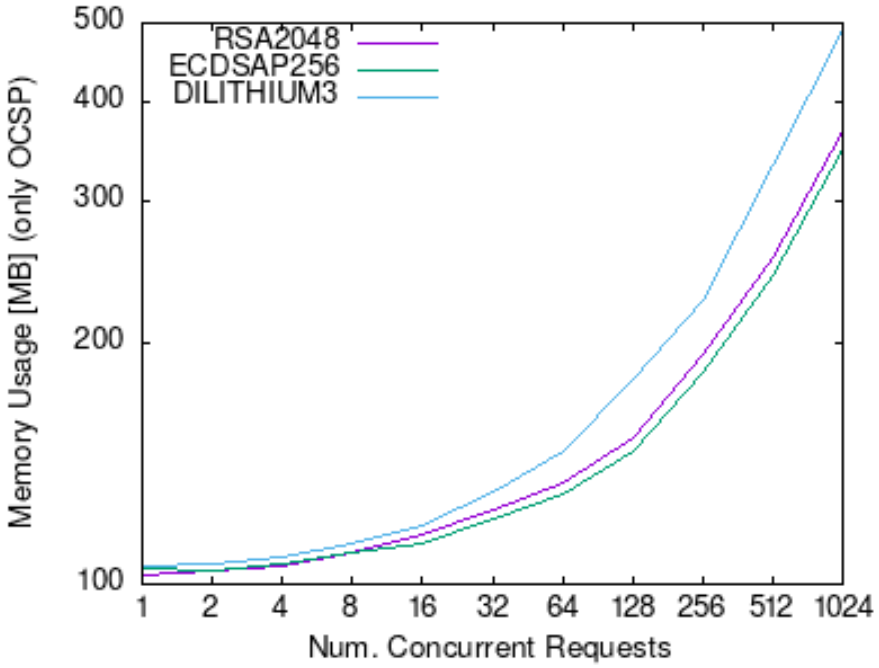


Figure 6.6: OCSP Sever Memory Usage (MB) for RSA, ECDSA and Dilithium

Algorithm	TPS	CPU(%)	Bandwidth(Mbps)	Memory(MB)	Latency(ms)
RSA	2124	92	23	255	240
ECDSA	5743	90	36	242	88
Dilithium	3934	91	284	333	129

Table 6.2: Maximum performance results (at 512 concurrent requests), for RSA, ECDSA and Dilithium

maximum corresponds approximately at 512 concurrency.

The Table 6.3 shows the rates (Results/concurrency) in the first region (the linear region), below 4 concurrency. The rates are seen for TPS, CPU and Bandwidth only. Table 6.4 shows the results for the Memory and Latency below 4 concurrency, as the values are almost constant.

Finally, the Table 6.6 gives the maximum resources needed for the database and the client, at 512 of concurrency. We can see that neither the database nor the client exceed the 50% of CPU. This means that the resources of the client and the database VM are not fully used in

	TPS/conc.	CPU(%)/conc.	Bandwidth(Mbps)/conc.
RSA	145	10.5	1.6
ECDSA	237	9.5	1.5
Dilithium	211	9.75	15

Table 6.3: CPU, TPS and Bandwidth rates below 4 concurrent requests

	Memory(MB)	Latency(ms)
RSA	105	8
ECDSA	106	4
Dilithium	107	5

Table 6.4: Memory and Latency values below 4 concurrent requests

	Latency(ms)/concurrency
RSA	0.45
ECDSA	0.17
Dilithium	0.25

Table 6.5: Latency rate above 4 concurrent requests

any case.

6.3 Conclusions

Comparing the similarity of results of CPU Usage with the TPS, it can be concluded that the TPS is strongly related to the CPU usage. It can be seen how the more the server is stressed with OCSP requests, the more resources the computer needs in order to sign the OCSP responses. With low concurrency the CPU is capable of handling the various requests distributing them between the cores, but as around 4 concurrent requests, it cannot handle all of them at the same rate, and the TPS rate starts decreasing, until it stabilizes at some maximum throughput. Signature verification is done by the client, but looking at the CPU usage of the client, we see that it consumes less CPU, never reaching its maximum, and same happens for the database CPU usage. So it can be concluded that signing is the most resource intensive operation, and it's what defines the TPS. Comparing the three algorithms, same behaviors but different results can be seen. Dilithium gives a maximum of 3900 TPS, which supposes 40% less than ECDSA but 50% more than RSA. The TPS rates also gives results in the same direction, but not so acute, where Dilithium performs 10% less than ECDSA, but 30% more than RSA, resulting in very acceptable results for Dilithium referring to its CPU usage.

Latency results are also strongly related to the TPS, but the behavior is inverse, being

	RSA	ECDSA	Dilithium
Client	23	40	48
Database	19	30	24

Table 6.6: Maximum CPU (%) Usage at 512 concurrent requests for the client and the database

flat at the beginning but as the TPS rate starts decreasing at around 4 concurrency. The latency, which until that point was constant below 10ms, starts increasing at constant linear rate. Mainly because the server must handle more and more parallel requests at the same time. The latency rate of each algorithm gives is different, Dilithium Latency increase rate is 45% less than RSA, but compared to ECDSA is 30% higher.

Bandwidth behavior is also similar to the TPS and CPU, increasing linearly at the beginning and, above 4 concurrency, slowly decreasing the rate until reaching a maximum. Looking at the bandwidth usage, it can be seen that here is where Dilithium differs from other algorithms. This difference is mainly because the signature sizes and key public key sizes, that must be sent through the network. Here Dilithium needs around a factor of 10 more bandwidth than other algorithms. Increasing at the beginning at a rate of 15Mbps/concurrent requests and reaching the maximum at almost 285 Mbps. Of course, this limit is reached because the limit resources of the PC where reached, in case of more resources, more bandwidth would be needed. In this case paying special attention at the Bandwidth is needed as it could suppose a bottleneck.

Finally, we see how the memory usage is at the beginning flat, but as TPS rate decreases, the memory usage increases exponentially, mainly because the memory resources are not freed at the same speed that they are requested. The case of the Memory needed is also a bit special for the case of the Dilithium, as the increase in memory is exponential. Dilithium memory Usage increases at a faster rate, and in some case it could happen that the memory needed reaches the maximum. Reaching the maximum memory is really critical and even could cause the server to stop running.

Appendix A

Examples parsed with OpenSSL

A.1 OCSP Request

```
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: 99F14D28B633E261C6D08AFA0FA8EAD63715EE39
          Issuer Key Hash: C8B89D0F95B44F0ECCE8A54978553C5EE7172704
          Serial Number: 12BBD2B91A3BBA326649C48BC235B1FF
```

A.2 OCSP Response

```
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: 93D5D335BA0AC607783A01555B7D07A7DF66F1C7
    Produced At: Feb 19 15:27:49 2022 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 99F14D28B633E261C6D08AFA0FA8EAD63715EE39
      Issuer Key Hash: C8B89D0F95B44F0ECCE8A54978553C5EE7172704
      Serial Number: 12BBD2B91A3BBA326649C48BC235B1FF
```

```
    Cert Status: good
    This Update: Feb 19 15:27:49 2022 GMT
    Next Update: Feb 19 23:27:49 2022 GMT
        Response Single Extensions:
            OCSP Archive Cutoff:
                Aug 20 14:53:11 2021 GMT

    Signature Algorithm: sha256WithRSAEncryption
         27:98:cd:fe:94:37:6b:ad:71:47:7f:c1:bf:6d:9b:f1:e1:51:
         ed:64:80:8e:d0:59:62:d5:18:d5:de:91:f1:0a:c6:f1:2b:ef:
         80:eb:90:a4:0f:e7:32:38:92:9f:d6:09:92:ff:a5:a1:af:c3:
         ec:a0:9b:8e:db:ba:15:57:cf:c5:59:db:c2:6c:c4:29:1e:6c:
         50:1d:4d:07:f8:35:07:86:1a:4d:5b:b0:5c:de:c7:b1:d3:22:
         cd:39:8b:2d:97:70:e2:4e:d5:54:a2:04:42:52:17:ee:03:dc:
         5c:c3:1f:9c:48:15:58:96:f7:3d:79:71:89:0a:5f:22:9b:01:
         d7:0a:82:a8:c4:e9:0b:b6:da:10:cd:e4:2a:f5:db:ba:98:57:
         64:fc:2b:9d:98:1e:0d:ae:f5:aa:38:08:de:6b:6c:14:e7:3a:
         58:76:3a:22:20:a5:7d:03:b0:e3:2a:42:49:78:31:cc:5f:93:
         0f:de:9f:31:f7:96:4e:b5:75:5e:58:6d:a2:62:6d:6d:da:60:
         46:7a:b8:a9:d8:1a:99:94:f1:01:cc:54:52:12:a4:57:c2:a4:
         f2:c3:a9:e6:d0:09:ca:78:63:ae:b8:e5:2d:87:9f:45:ad:cb:
         d1:b9:46:a3:19:1f:93:68:b2:43:84:bc:f2:1e:f3:8f:c4:d8:
         7a:f5:4c:45
```

A.3 VA Certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            29:0d:84:70:3b:dc:61:89:a5:e7:f9:e6:80:89:2d:7f
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Root User CA 1
        Validity
            Not Before: Oct  6 16:53:26 2021 GMT
            Not After : Oct  6 16:53:23 2022 GMT
        Subject: CN=VA 1
```

```
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
        Modulus:
            00:e0:74:41:5a:de:b1:69:1d:ff:0e:7f:47:f6:cd:
            bc:26:82:e9:98:de:e2:ce:80:a7:b0:a5:0c:45:69:
            a6:ce:fd:eb:b1:10:a1:f0:cc:3c:98:49:4b:94:15:
            32:dc:8a:8c:2e:fc:36:92:04:f6:16:8f:e5:6e:ac:
            d6:44:87:b6:39:63:80:1a:47:7d:bb:6d:45:ec:14:
            d4:c0:62:71:55:61:f6:28:82:44:ba:4b:b5:4e:65:
            9a:23:69:24:04:d2:d6:78:68:9a:2c:bc:23:97:df:
            f1:b2:53:e0:26:08:c2:2f:f7:8a:9b:28:3a:70:61:
            35:69:a6:7b:76:e3:aa:94:43:a3:1c:8a:c0:e3:40:
            52:13:e0:ed:d3:29:ad:5e:ef:2f:81:71:bd:7a:e5:
            68:b9:d6:16:cf:1c:c8:98:e5:f0:74:9d:49:98:ca:
            06:e6:9e:62:8b:d1:24:41:78:da:e7:aa:1f:d0:ca:
            f2:db:90:ef:4a:47:e6:63:99:da:4f:06:37:75:9c:
            70:a8:69:3b:46:44:aa:a5:cb:bf:bd:42:24:e0:9d:
            c3:04:2f:e1:2a:41:9d:5a:35:b2:d5:c0:88:e6:68:
            7c:64:68:a1:9b:a8:a7:b3:b7:98:ab:4e:4b:c0:55:
            32:4a:6b:f2:83:ac:db:fb:44:32:1a:48:0e:3d:22:
            6d:2f
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Subject Key Identifier:
        93:D5:D3:35:BA:0A:C6:07:78:3A:01:55:5B:7D:07: \\
                A7:DF:66:F1:C7
    X509v3 Authority Key Identifier:
        keyid:DB:81:57:07:3A:0E:64:77:07:29:77:E7:D4: \\
                CF:85:B4:F2:92:27:81

    X509v3 Key Usage: critical
        Digital Signature
    X509v3 Extended Key Usage:
```

```
            OCSP Signing
        OCSP No Check:

Signature Algorithm: sha256WithRSAEncryption
     35:5a:ac:e6:c8:34:70:72:28:7a:39:28:8e:c6:86:33:0c:6c:
     3b:34:54:66:e4:7d:1e:e9:6f:91:b3:e8:e7:81:9d:38:d2:5f:
     08:88:ad:4a:00:ee:7b:6b:f5:2a:29:59:7c:14:32:17:38:b6:
     e0:a7:15:70:5c:30:18:94:db:2a:c1:ef:51:e4:bc:6d:e6:32:
     67:65:d5:b4:77:d6:1c:64:ef:ee:88:44:cb:67:2d:9c:b5:f2:
     7d:c1:61:77:20:36:fb:d7:9a:46:46:fc:62:3b:e8:3c:03:b8:
     60:a2:a3:82:30:a1:d8:40:5c:50:8a:56:63:23:fb:7d:7d:f6:
     35:31:7c:ff:6d:bb:6d:9a:03:4d:21:64:5d:cf:06:e3:0b:58:
     08:e7:da:1b:f3:1b:9d:4b:cd:44:4f:a9:d2:8c:64:cb:80:72:
     89:27:cf:d5:55:f4:a7:6d:67:0a:42:b7:33:3b:44:c6:a7:27:
     9d:de:d3:68:b8:8b:39:a9:62:3c:a1:b0:03:33:0f:3b:14:de:
     f4:d9:f9:73:58:1f:63:38:f5:20:86:71:c5:df:68:ac:92:e0:
     a4:65:2a:63:e2:8c:6e:1d:21:3e:ae:e9:ce:f4:67:ee:b5:32:
     a7:46:1a:83:43:f3:5e:3a:73:c1:e3:b9:ee:fc:e5:e7:c3:c1:
     f5:0c:7b:29
```

Bibliography

[1] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2003.

[2] Antoine Beaupré. Roca: Return of the coppersmith attack. Technical report.

[3] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549:188–194, 2017.

[4] Ward Beullens. Breaking rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Paper 2022/214, 2022. https://eprint.iacr.org/2022/214.

[5] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[6] Johannes Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. 01 2013.

[7] Sofía Celi, Armando Faz-Hernández, Nick Sullivan, Goutam Tamvada, Luke Valenta, Thom Wiggers, Bas Westerbaan, and Christopher A. Wood. Implementing and measuring kemtls. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology – LATINCRYPT 2021*, pages 88–107, Cham, 2021. Springer International Publishing.

[8] ETSI CYBER Quantum Safe Cryptography. Tr 103 616 v1.1.1, cyber: Quantum-safe signatures crystals-dilithium. Technical Report DTR/CYBER-QSC-008, ETSI, 2021. algorithm, cybersecurity.

[9] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium, algorithm specifications and supporting documentation (version 3.1).

[10] Farnoud Farahmand. Implementing and benchmarking seven round 2 lattice-based key encapsulation mechanisms using a software/hardware codesign approach. 2019.

[11] Oriol Farràs. Lecture notes on cryptography and blockchain technology unit 10,11- digital signatures. Technical report, Universitat Rovira i Virgili, 2022.

[12] Oriol Farràs. Lecture notes on cryptography and blockchain technology unit 15- quantum-resistant cryptography. Technical report, Universitat Rovira i Virgili, 2022.

[13] Vatistas Kostalabros, Jordi González, Oriol Farras, Miquel Moreto, and Carles Hernández. Hls-based hw/sw co-design of the post-quantum classic mceliece cryptosystem. pages 52–59, 08 2021.

[14] K.M. Martin. *Everyday Cryptography: Fundamental Principles and Applications*. Oxford University Press, 2017.

[15] MATZOV. Report on the Security of LWE: Improved Dual Lattice Attack, April 2022.

[16] NIST. Post-quantum cryptography, round 3 submissions.

[17] Mike Ounsworth and Massimiliano Pala. Composite Signatures For Use In Internet PKI. Internet-Draft draft-ounsworth-pq-composite-sigs-06, Internet Engineering Task Force, February 2022. Work in Progress.

[18] Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners.* Springer, 2010.

[19] Chris Peikert. A decade of lattice cryptography. *IACR Cryptology ePrint Archive*, 2015:939, 2015.

[20] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.

[21] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[22] Alexander Truskovsky, Daniel Van Geest, Scott Fluhrer, Panos Kampanakis, Mike Ounsworth, and Serge Mister. Multiple Public-Key Algorithm X.509 Certificates. Internet-Draft draft-truskovsky-lamps-pq-hybrid-x509-01, Internet Engineering Task Force, August 2018. Work in Progress.