Università di Pisa

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Fisica

Anno Accademico 2013/2014

Elaborato Finale

Post Quantum Cryptography

Candidato:

Antonio Vaira

Relatori:

Prof. Oliver Morsch

Ing. Cristiano Borrelli

Alla mia mamma

Abstract

I started my experience with cryptography within the Airbus environment working on this master thesis. I have been asked to provide a framework, or in other words, a big picture about present-day alternatives to the most used public key crypto-system, the RSA, that are supposed to be quantum resistant. The final application of my work eventually resulted in recommendations on how to handle the quantum threat in the near future. This was quite a complex task to accomplish because it involves a huge variety of topics but by physical background was really helpful in facing it. Not because of specific and previous knowledge but for the *mathematical tools*acquired during the studies and especially for that attitude that belongs to out category that make us, physicist *problem solver* in a large variety of fields.

Indeed I also tried to go a bit further with my studies. I took one of the most promising algorithm on my opinion, but not well known yet so unfeasible for a recommendation and therefore an implementation in the close future, and I tried to figure out how to enhance it from both a security and an operational point of view (how to increase the correctness ratio of the decryption and the speed of the cryptographic operations). It followed a period of time in which I improved my skills with few computing languages and in the end I decided to implement a *toy model* at a high level using an interface that already implements all the mathematical and algebraical structures used to build the model. This interface is Sage [62], in particular I used the version 6.4.1, see the appendix for the implementation of the cryptographic functions. The interface can be programmed using a ”Python-style” language and it is a very high level interface, i.e.the underlying functions are considered like black-boxes without needing to know the exact behaviors of every and each function.

Acknowledgements

I would like to thank my boss and advisor *Cristiano Borrelli* together with *Helmut Kaufmann*, my tutor inside the Airbus Group, for the great opportunity of working together with them and for all the support and advices they gave me during my master thesis preparation.

Special thanks go to my tutor, from the university of Pisa, *Oliver Morsch*, for his invaluable help and patience, especially in the reviewing phase. Without his help my master thesis would not look at all as it is now.

I would like to say a big thank you also to my colleagues, form the *Airbus Group Trust Center* in Munich, for the wonderful time we spent together and the continuous exchange of point of views and hints about the current state of the art cryptography.

Last but not least that deserve a huge thank you is my family, with a special mention to my mother, for both the economical and spiritual support along my entire study path.

Contents

1 Introduction **4**

1.1 What does physics add to the current scenario? 7

1.2 What exactly is post-quantum cryptography? 11

2 Current Public Key crypto-systems and the Quantum Threat **15**

2.1 From the origins to a Public Key Infrastructure 15

2.2 The R.S.A. algorithm 19

2.3 Quantum Computer and Quantum Algorithms 21

2.3.1 Quantum Computer 21

2.3.2 Shor's algorithm 24

2.3.3 Grover's algorithm 27

3 Mainstream alternatives to the current PKC **29**

3.1 Code-based crypto-systems 30

3.1.1 McElice crypto-system 30

3.1.2 Niederreiter variant 32

3.2 Lattice-based crypto-systems 33

3.2.1 The GGH/HNF Public Key crypto-system 35

3.2.2 NTRU crypto-system 36

3.2.3 LWE-Based crypto-system 38

3.3 Multivariate Public Key crypto-systems 40

3.3.1 Basic Constructions 41

3.4 Framework 43

3.4.1 Algorithmic Problems 44

3.4.2 Quantum Algorithm Weakness 45

3.4.3 Attack model vulnerabilities 45

3.4.4 RSA-style signature 46

3.4.5 Dimension and complexity 46

3.5 Reasons to choose the lattice based PKC 46

4 A closer look at the lattice based PKC 48

4.1 Introduction 48

4.2 The GGH crypto-system 48

4.2.1 Keys generation 49

4.2.2 Encryption and Decryption 50

4.3 The NTRU crypto-system 52

4.3.1 Keys generation 53

4.3.2 Encryption and Decryption 55

4.4 The Learning with Errors crypto-system 56

4.4.1 Keys generation 57

4.4.2 Encryption and Decryption 58

5 LWE Variants 60

5.1 Crypto-system Variants 60

5.1.1 Regev's crypto-system 61

5.1.2 Enhanced Regev's crypto-system 61

5.1.3 LWE Ring Variant 62

6 Personal Implementation of the ring-LWE PKC 64

6.1 Enhancements/Modifications to the original Ring-LWE 68

7 Conclusions 79

A Scripts 82

A.1 key-pair generation 82

A.2 encrypt plus hamming encoding 84

A.3 decryption 89

Chapter 1

Introduction

The relationship between physics and cryptography is not so obvious, at a first sight, but it is absolutely out of question that physics gave a big contribution to the development of cryptography, starting from the first examples in the human history until the most recent breakthroughs in quantum information that provides a tool to both break and/or strengthen the effectiveness of cryptography, [33, 16].

In this introduction I will briefly address the relationship between physics and cryptography in order to set a framework to use as starting point for the development of my whole thesis.

The invention of cryptography can be approximately dated around the same period as the invention of writing, therefore it underwent to an impressive evolution since its origins. One of the first, and well documented, examples is the Greek *scytale*: a tool used to perform a transposition cipher. Transposition ciphers encrypt plaintext by moving small pieces of the message around. Anagrams are a primitive transposition cipher [42]. The encryption of a message, using the scytale was merely based on rolling a long stripe of parchment around a baton and then writing a message along the direction of the baton (see picture1.1). The message is then hidden simply unwrapping the stripe. To decrypt the message the stripe needs to be rolled on another baton, with the same dimensions

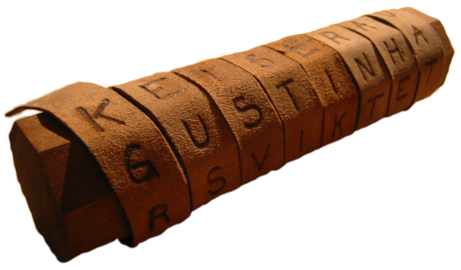


Figure 1.1: The Greek Scytale: one of the first example of cryptographic application.

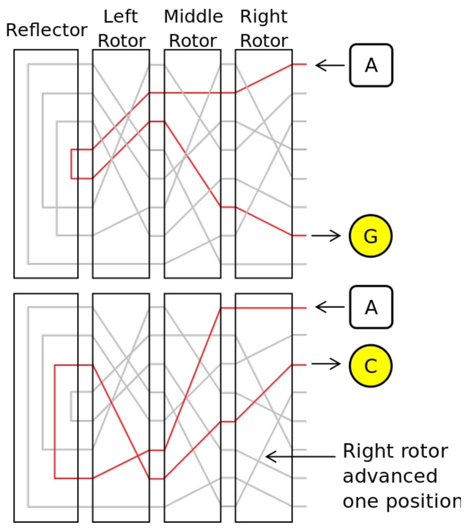


Figure 1.2: An example of the scrambling action of the rotors in the *Enigma machine*.

as the previous one. The decryption was supposed to be not possible without being in possession of the right baton.

Another example, closer to our period, is the *Enigma machine*: used by the German government starting from the 1920 and during the Second World War by the Nazi army. Enigma is based on a *polyalphabetic substitution cipher*[1]. It is basically an electro-mechanical machine made-up of a keyboard; a set of rotors placed on the same pivot; a stepping mechanism that turns one or more rotors of one twenty-sixth of a whole round; twenty-six lamps with a letter printed on each of them. The mechanical parts of the machine form a circuit that is closed each time a key is pressed, and the resulting current flow originated from a battery follow a certain path that finally hit a lamp lighting it. Each time a key is pressed one or more rotors make a step creating a different path for the current flow. The continuous changing of the rotors configuration ensures the security of the machine, i.e. the same latter is encrypted in two different ways after two consecutive digitations, see picture 1.2. To decrypt a ciphertext obtained from an *Enigma* machine another one is needed and it has to be set with the same initial starting conditions, we would say in modern terminology: ”*with the same seed*”.

Those are just two examples of cryptographic implementations but history indeed provides a plethora of other examples. See in the literature for example the Caesar cipher or the Vigenère one. Many implementations of ciphers were simple substitution ciphers, therefore much less secure than the polyalphabetic counterpart.

[1]In the polyalphabetic substitution ciphers the encryption is simply the substitution of a letter with another. Here ”poly” means that multiple encryptions of the same letter produce different outputs. The early ciphers didn't have this ”feature” and therefore they were far more easier to break.

From this scenario cryptography evolved toward a more abstract formulation, [42]. The first implementations aimed to encrypt/decrypt information coded with a natural language, i.e. words and sentence encrypted via a substitution or transposition techniques. On the other side the more recent implementations allows to encrypt/decrypt *information* in a much broader sense, a simple example is: words, pictures and anything else that can be coded with bits. Even more important, modern cryptography allows one not only to encrypt but can be used to enforce the identity proofing and the digital signature[2]. These capabilities are fully exploited by daily basis activities that we all perform using the internet, like home-banking, e-commerce or e-vote, etc. In the next sections I will briefly explore the state of the art cryptography algorithms with particular attention to the asymmetric crypto-system, like the *RSA*.

Older than the asymmetric crypto-system are the symmetric ones. In general one considers as symmetric all the algorithms that are based solely on sharing a secret. We can include in this category all the algorithm developed during the history (also the scytale and the Enigma machine belong to it). This way of performing encryption is now referred to as "symmetric".

The general working principles, valid for all the implementations, can be summarized in the following steps.

A plain-text m is encrypted using a so called *secret key* k, that is nothing more than a secret rule/information used to hide [3] the message but preserving its structure in order to keep it recoverable:

$$c = f_k(m) \tag{1.1}$$

The encrypted plain-text can be recovered only if the secret key is known:

$$m = f_k^{-1}(c) \tag{1.2}$$

In the first implementations of the symmetric encryption algorithms there is no distinction between the private key and the encryption algorithm itself because they are the "same". The secret key was considered the collection of rules used to encrypt a message

[2]Digital signature is a cryptographic scheme for demonstrating the authenticity of a digital message or document. Furthermore the digital signature is used to enforce the non repudiation of the message (i.e the sender of the signed message cannot deny having sent the message) and to guarantee the integrity of the sent message(i.e. it was not altered during the transmission).

[3]Krypto from the Greek means to hide

and that you need to follow with the inverse order to decrypt the ciphertext, that is also the definition of algorithm.

In the latest years there was a general understanding that the algorithm and the key need to be separate. In this way the security of the encryption mechanisms doesn't need to rely on the algorithms anymore, that from now on can even be disclosed, but only on a simple secret. The disclosure of the algorithms is an important point: it allows the creation and standardization of protocols and a continuous crypto-analytic work on the algorithms themselves in order to patch all the weakness [7, 8, 28].

Anyway the symmetric encryption scheme has a big weakness: there is no obvious way of exchanging the secret key in an electronic way. This is an issue that can be circumvented with the usage of the asymmetric cryptography, as we already do, or with the help of quantum physics [47], as will be explained in the following section.

1.1 What does physics add to the current scenario?

In the past twenty years cryptography didn't evolve so much anymore. The last big contribution can be dated back to the invention of the RSA crypto-system, an asymmetric crypto-system based on the intractability of solving the underlying algorithmical problem. This is one, of the not so frequent, examples in cryptography where the security of the algorithm is based on a mathematical problem rather than an heuristic approach.

The *cryptographic primitives* [4] are usually designed ad-hoc, according to the purpose they are supposed to accomplish. During the design process the crypto-system regularly undergo crypto-analytic [5] studies until it is considered secure and reliable, in a way we can see it as a "trust building process".

This is not the case for the RSA, where indeed the security of the crypto-system is mainly based on the algorithmic difficulty of finding the n^{th} root of a number modulo a composite N. For large N no efficient algorithmic solution is known. The hardness of this problem can be in turn reduced to the hardness of factoring big numbers, like the composite $N = p \cdot q$ for p and q prime numbers.

It is exactly in this scenario that quantum physics will be a "major player" in the

[4]Cryptographic primitives are well-known and trusted cryptographic algorithms used to build cryptographic protocols.

[5]The crypto-analysis is the study carried out on a primitive with the purpose of breaking it, either mounting a theoretical (attack against the "math" of the problem) or a side channel [7] attack (against the hardware/software implementation).

next future of cryptography. For the time being there are two main[6] applications of the quantum physics in the field of cryptography:

- **Quantum Computers**,
- **Quantum Cryptography**.

Even though the development of a fully operational quantum computer is still uncertain [53], there are already two quantum algorithms that affect the security of the current state of the art crypto-system. A more detailed description will be provided in the following sections, just to give a glimpse: the Shor's algorithm [19, 33] is a quantum algorithm for number factorization. It is supposed to factorize numbers in a polynomial time respect to the size of the number that is factorized. The best currently known algorithms are able to factorize number in an exponential time [42]. Having an efficient algorithm for factorization would completely break the security of the asymmetric primitives used today because their security is based on the difficulty of such operation, where difficulty means a large cost in terms of time/resources, eventually more than the ”value” of the information intended to retrieve.

For the time being a fully operational quantum computer seems still far away [53]. To run the Shor's algorithm for a number consisting of thousands of bits we would need a quantum computer with a registry of thousands of qubits [33]. All the experiments carried out until today didn't manage to build a quantum computer with such a big registry. The main obstacle comes from to the difficulty of keeping a large number of qubits in a correlated state over a long time.

On the other hand there is the quantum cryptography, or to be more accurate: the *quantum key distribution* (QKD) [47], a countermeasure that can face the threat that a quantum computer poses especially against the ”RSA-like” crypto-system. Its implementations can exploit either the *quantum superposition* or the *quantum entanglement*.

For example in the first implementation of a QKD protocol, like the *BB84*, the bits 0 and 1 can be encoded in two different bases, a rectilinear and a diagonal one. In the rectilinear basis the bit 0 will be encoded as the quantum state : $|\uparrow\rangle$, the bit 1 will be $|\rightarrow\rangle$; in the diagonal basis 0 will be $|\nearrow\rangle$ and 1 will be $|\searrow\rangle$. These basis are usually built using the polarizations of photons like in the picutre 1.3.

The key exchange will take place between two parties: A and B. As first step A generates a sequence of random bits and randomly encodes each of them in one of the

[6]Another interesting application would be the generation of random numbers, that is one of the most critic aspects of the practical implementations [42]

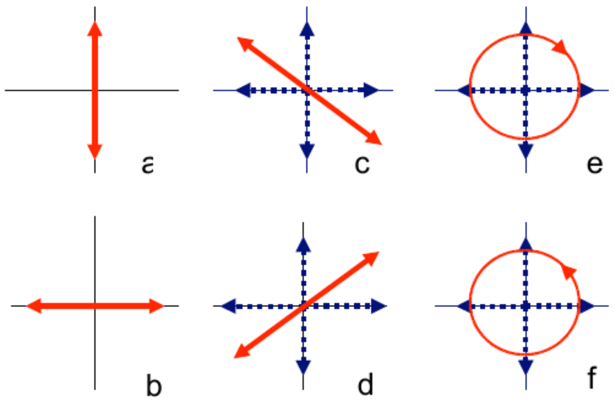


Figure 1.3: Examples of rectilinear basis (a,b), diagonal basis (c,d), circular basis (e,f).

$A\, basis$	$+$	$+$	$\times$	$+$	$\times$	$+$	$\times$	$+$	$+$
$A\, random\, bits$	1	0	0	1	1	0	0	0	1
$A\, photons$	$\lvert\rightarrow\rangle$	$\lvert\uparrow\rangle$	$\lvert\nearrow\rangle$	$\lvert\rightarrow\rangle$	$\lvert\searrow\rangle$	$\lvert\uparrow\rangle$	$\lvert\nearrow\rangle$	$\lvert\uparrow\rangle$	$\lvert\rightarrow\rangle$
$B\, basis$	$+$	$\times$	$\times$	$+$	$+$	$\times$	$\times$	$\times$	$+$
$B\, recovered\, bits$	1	0	0	1	0	1	0	1	1
$Same\, basis?$	YES	NO	YES	YES	NO	NO	YES	NO	YES
$Sifted\, key$	1		0	1			0		1

Table 1.1: Example of usage of QKD protocol.

two bases. Since the two bases are not orthogonal if one photon is measured in the wrong basis it will produce a random output. After the generation/encoding procedure A will send the encoded bits over the quantum channel and B will measure each of them in a random basis. A can then communicate to B, over the classical channel, which basis she used to encode each bit so they will discard the bits that were recovered by B using the wrong basis. The table 1.1 represents a simplified implementation of the protocol.

Another famous implementation of a QKD is the *Ekert's* one, [47]. It exploits the quantum entanglement of a pair of photons generated in the same physical process and then distributed to A and B. Due to the entanglement both A and B will get the same result when they will measure if the polarization of a photon is either horizontal or vertical and furthermore the sequence of measured value is absolutely random, it means that there is no way neither for A nor B to predict the key.

In such scenario, with QKD on the playground, a secure communication between two parties would need two channels, a classic one and a quantum one. The quantum channel can be used to securely distribute the keys of a classic symmetric cipher(see details in the next sections). The classic channel can then be used to exchange the ciphertexts. Notice that there is no security requirement for any of this channels.

The secrecy of the keys distributed with a QKD protocol is ensured because a measurement on a quantum information would modify it, this property makes any eavesdropping attempt detectable, therefore a simple countermeasure would be dropping the channel and opening a new one for a secure keys exchange. After the key exchange phase the two parties are in possession of the same secret key that can be used to encrypt a message using for example the simple *one-time* padding scheme.

The one-time padding scheme is the only cipher that is proven to be unbreakable as long as each message is encrypted with a key that is of the same length of the message and is also truly fresh and random [42]. The algorithm is astonishingly simple: both the message and the key are expressed in bits and they are pairwise added modulo 2. This operation is also known as pairwise XOR[7]. It is understandable how impracticable such encryption algorithm would be in everyday life and with the current technology. It is impossible to share the secret key over the internet or over any other channel that doesn't involve a physical protection: like a guarded courier.

A historical aside: the one-time pad was used as encryption scheme to protect the communication between Washington and the Moscow during the Cold War. The key

[7]The XOR operator take as input two bit and output a zero if the inputs are the same and a one if the inputs are different.

distribution was performed by a courier who transported a suitcase with secret random keys each time it was needed.

So if on one side quantum physics seems to represent the ”end of an era” for asymmetric encryption algorithm, on the other side seems to propose also a solution with ”quantum cryptography”. But it is not as easy as it seems. At the current level of technology there are severe limitations to the massive usage of quantum key distribution. For example the few optical fibers deployed are not suitable for transmitting entangled photons or maintaining the photons' polarization over long distances. Even more difficult: it is essential to have and end-to-end optical connection to perform QKD, otherwise the security assumptions would fall. From a technological point of view there is still a severe limitation coming from the extreme difficulty of producing single photons. So far they are produced with highly attenuated lasers but that in turn can also produce couples of correlated photons. They are easily exploited to mount an attack, since a measurement on one photon of the couple doesn't reveal an alteration when the other photon is measured, and recover the secret key entirely or partially.

It is exactly at this point that *post-quantum cryptography* [10] comes into play. We cannot afford to underestimate the progresses made in the realization of quantum computer, we need an alternative in case the quantum computer will become reality[8] in the next decade or so or before quantum cryptography will be reliable.

1.2 What exactly is post-quantum cryptography?

Post-quantum cryptography is the study of the cryptographic primitives that are supposed to be unbreakable by a quantum attack. It is pretty difficult to address the question whether a cryptographic algorithm is weak against a quantum computer. So far only two quantum algorithms are exploitable against the current state of the art of cryptography: Shor's algorithm and Grover's algorithm [4, 25](for an explanation on their basic working principles see the next sections). Grover's algorithm is capable of boosting the searching speed in an unsorted database. This is an algorithm that can potentially affect all crypto-systems, for instance it can boost a brute force attack [8], i.e. an attack that looks for a key, recursively trying all the possible combinations until the right one is found. This attack can be taken into account merely increasing the key sizes in all the crypto-systems.

[8]The company *D-wave* claimed to have built a quantum computer with a registry of 128qubits. Either this is true or not it is not yet a threat for the RSA but it will be a serious threat when the registry will approach a thousand bits size.

Shor's algorithm is a more persistent treat, since it breaks the mathematical problem on which the asymmetric crypto-systems are based: i.e. make the factorization easy.

Classically the factorization problem is considered a hard problem. This definition is not so clear and actually is pretty unpractical making comparisons between the difficulties of several algorithmic problems. Therefore in the theoretical study of the algorithms is necessary to introduce the concept of *complexity class*. The theory is quite involved but in order to give a glimpse we can say that problems supposed to have the same difficulty are in the same complexity class. The algorithmic problems in the same class are all reducible to one algorithmic problem of the same class with an algorithm running in polynomial time.

The most fundamental classical classes are *P* and *NP* [1]. The *P-class* contains all the decision problems that can be solved by a deterministic Turing machine using a polynomial amount of resources, in other words this class contains the algorithmic problems that are easily solvable. A Turing machine is a theoretical machine that read and write symbols, one at time, on an endless tape following a set of rules[9]. The *NP-class*, where NP stands for *non-deterministic polynomial*time, and contains all the decisions problem in which a "yes" instance can be accepted in polynomial time by a *non-deterministic* Turing machine. A non-deterministic Turing machine may have a set of rules that prescribes more than one action for a given situation. This definition can be simply seen as the class of problems that might be hard to solve. The class of P problems is contained in the NP-class but the NP-class is supposed to be larger than the P-class. The factorization problem is inside the NP-class but outside the P-class, therefore there is no classical algorithm capable of factorizing a number in polynomial time, i.e. efficiently.

Quantum computation introduced another complexity class: the *BQP-class*. BQP stands for *bounded error, quantum, polynomial time*. It is the quantum equivalence to the classical BPP, *bounded error, probabilistic, polynomial time*. The *P-class* is a subset of the *BPP-class*, since in the BPP are contained the problems that are solvable in polynomial time and with a certain probability. This is the class where most of the practical implementation of the algorithms are included. The presumed relationship between the classes is summarized in the picture 1.4. The factorization problem is supposed to be outside the *P-class* but definitively inside the *BQP-class*.

[9]For sake of completeness: the Turing machine is a mathematical abstraction. A Turing machine is a computing model consisting of a finite set of states. An infinite tape which symbols from a finite alphabet can be written to and read from using a moving head, and a transition function that specifies the next state in terms of the current state and symbol currently pointed to by the head.

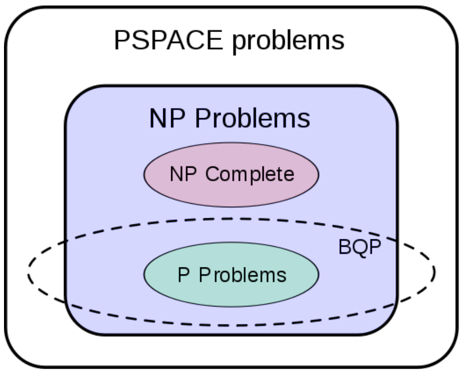


Figure 1.4: Complexity classes introduced with the introduction of the quantum computation.

At this point it is easy to understand that if we wish use cryptography, in the same way we do right now, all the underlying primitives need to be based on mathematical problems outside the *BQP-class*, even though the boundaries of this class may grow in the future making the distinction very hard.

In the post-quantum cryptography field there are already several proposals [10, 35] that base their security assumption on algorithmic problems outside the current *BQP-class* boundaries. The mainstream examples that I studied for this thesis are:

- the **code based crypto-systems** [41, 32] are encryption algorithms inspired to the telecommunication-engineering problem of recovering the correct signal transmitted over a noisy channel [22]. The signal is encoded with a particular ”code”, prior to the sending. This allows to correct up to a certain number of errors. Encoding a message is simply adding a certain number of parity/check bits to the message. A parity bit could be, for example, the sum modulo 2 of a couple of bits, therefore it helps in retrieving the original values of the bits if one error occurred in the communication.

  Following the principle of the noisy channel, a message can be encrypted adding to it an amount of artificial noise after the encoding. It can be decrypted only if particular information related to the used code are known.

- the **lattice-based crypto-systems** [56, 57, 45, 26] are encryption algorithms where the security is mostly based on the difficulty of solving certain problems in multi-dimensional lattices or on the difficulty of recognizing perturbed equations from unperturbed ones, that is in the same spirit of the code-based algorithm but in a lattice framework.

- **multivariate public key crypto-systems** [64, 52, 23] are encryption algorithms where the security is related to the difficulty of solving simultaneously a random set of quadratic equations. The encryption/decryption is mainly based on the evaluation of such equations at particular points.

After an initial study of these algorithmic families I tried to build a framework where to categorize and to find enough good reasons to choose one family of crypto-systems, first, and one particular crypto-system of this family, after, for a practical usage in a real world application. Finally in the last chapter I built a high-level implementation with some proposals for possible enhancements of one encryption algorithm.

In the following sections I briefly explain the basic working principles of the current public key crypto-systems and the two quantum algorithms exploitable in a ”quantum” crypto-analysis. Then I will give a glimpse of the working principles of the mainstream encryption algorithms that belongs to the three families already mentioned, with a particular attention to the *lattice-based* family and to the learning with errors encryption algorithms [51, 46, 15]. This is a special family of encryption algorithm built in a lattice framework, but based on the long-standing problem of *machine learning*, where the security relies on the difficulty of distinguishing, in a set of equations, which of them are perturbed and which are not. More explanations will be provided in the following sections. Finally in the last chapter I will show a personal implementation of the ring-LWE encryption, [39, 40, 21, 17, 37] algorithm introducing some modifications that could boost the encryption process and eventually enhance the security. To prove it I performed a multi-parameter comparison between my version of the algorithm and the one provided in the literature.

Chapter 2

Current Public Key crypto-systems and the Quantum Threat

2.1 From the origins to a Public Key Infrastructure

Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients. Modern cryptography is based not on sharing the decoding technique, the algorithm, but only a key, leaving the algorithm in the public domain and even standardizing it.

The main classical ciphers types are *transposition* ciphers, which rearrange the order of letters in a message and *substitution* ciphers, which systematically replace letters or groups of letters with other letters or groups of letters. A famous example, for the substitution cipher, is the Caesar cipher, in which each letter in the message is replaced by a letter at some fixed distance, in the alphabet, from the original. Suetonius reports that Julius Caesar used it with a shift of three to communicate with his generals.

The main problem about ciphertexts produced with a classical cipher always reveal statistical information about the plaintext, which can often be used to break them with the frequency analysis [28].

Essentially all ciphers remained vulnerable to the *frequency analysis* technique until the development of the polyalphabetic cipher attributed to Leon Battista Alberti around the year 1467. The innovation was to use different ciphers, in other words it is a substitution cipher that change the key at every given amount of letters.

In the 19th century it was finally recognized that the secrecy of a cipher's algorithm is not a sensible nor practical safeguard of message security, actually it was realized that any adequate cryptographic scheme should remain secure even if it is public. The secrecy of the key used should alone be sufficient to guaranty the security of cipher.

Finally in the 20th century appeared also "hardware" implementation of the cipher, that means automatic machine used to perform encryption/decryption, like the famous Enigma.

Cryptanalysis of the new mechanical devices became more and more both difficult and laborious. Due to this difficulty during the WWII, in the United Kingdom, was developed *Colossus*: the world's first programmable computer used for the decryption of ciphers generated by the German Army's machine.

After WWII the development of digital computers began. Not only was it of use for cryptanalysis but also in defining much more complex ciphers. Computers allowed the encryption of any kind of data representable in binary format: this allowed to move from the classical ciphers, that is linguistic cryptography to the cryptography as it is thought today, i.e. able to secure any kind of transaction.

Only in recent times there was a major breakthrough in the field: IBM personnel designed the algorithm that became the Federal Data Encryption Standard (DES, improved in the 3DES and AES), Whitfield Diffie and Martin Hellman published their key agreement algorithm, and the RSA algorithm was published in Martin Gardner's Scientific American column [59].

Since then, cryptography has become a widely used tool in communications, computer networks, and computer security in general. Eventually the implementation of the RSA, and the asymmetric encryption algorithms in general, culminates in a series of protocol that defines a *public key infrastructure* (PKI) [42]. All the protocols, of a PKI, aim to guarantee a high trust level on the issued credentials. In few simple words a PKI is the infrastructure that makes the RSA algorithm usable.

We use a PKI, in everyday life, several times per day, even outside a corporate environment. For example a simple authentication client/server can be handled via a PKI. We can use a PKI credentials also to digitally sign documents and of course to perform encryption.

An example of public key crypto-system usage can be sketched as simple encrypted transaction between to users A and B, see picture 2.1. For instance A decides to send an

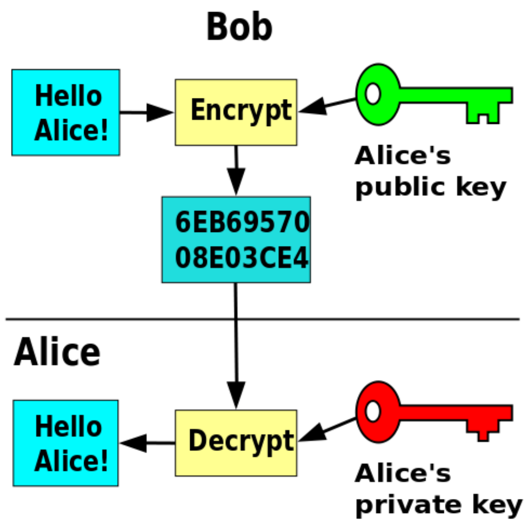


Figure 2.1: Sketched implementation of public key crypto-system usage.

encrypted message to B, she can encrypt a message using B's public key and send it to him. Only B can now recover the original message, because he is the only one in possession of his private key. Now B can reply to A sending her an encrypted message using her public key and so on. This schema is very simple and easy to use from a theoretical perspective but it has several weaknesses that have to be taken into account:

- assuming that there is a way of defining such keys with such relation, what is the definition of "hard" in the sentence "it is supposed to be hard to recover the private key from the public key" and how hard is it to recover the private key from a sample of previously chosen encrypted messages;
- it is vulnerable to impersonation attack, that means a malicious attacker C can claim to be A when she is speaking with B and vice versa. In this way she can submit her public key to both A and B and in turn they will believe to talk to A or B ignoring C.

The latest point can be easily accounted defining the so called *Certification Authorities*, CA [42]. The CA is supposed to certify the identity of a user that will participate to a possible transaction between all the certified users and bind the identity to a public key. The process of the identity certification is defined by the implementation requirements. The binding of the public key to an identity take place in the creation of a certificate defined by the *X.509* ITU-T[1] standard. The CA is therefore one of the most expensive

[1]ITU-T stands for International Telecommunication Union, division for telecommunications.

and important part of a PKI. A CA stores several sensitive data and one of its duty is to produce a list of valid certificates and update it according to the policy under which it operates. Therefore it is natural to think of a CA as a highly available and redundant system[2]. The PKI protocols and the CA, on the other hand, are not related to the practical implementation of the crypto-system employed in the PKI itself, therefore they will survive to a crypto-system "upgrade" toward the post quantum ones.

The answer to the first question, as already depicted in the introduction, is a bit more delicate. As should be already clear to the reader, the concept of security, in particular when we speak about encryption, does not make so much sense if we keep the time out of the discussion. In general we can substitute the word "hard" with "time consuming" in all the sentences that speak about the hardness of a crypto-system.

Breaking a crypto-system means either recovering a message from the ciphertext or recovering the private key from the public key. This is known as total break. There are several attack models used to evaluate the security of a practical implementation, like the *chosen plaintext attack* (CPA), *chosen ciphertext attack* (CCA) or *adaptive chosen-ciphertext attack* (CCA2). In order to keep the discussion at a high level I will not speak about them in my thesis, they will only be briefly mentioned in the following sections. More information can be found in the literature [5].

So far the cryptography had an "easy life" and all the crypto-systems employed could be safely considered secure, the security flaws are usually caused by wrong implementation or incorrect usage of the protocols. But if the realization of a large quantum computer[3] will take place all the current public key crypto-systems used today will no longer be secure and will be needing a replacement, on the contrary the symmetric encyption and the secure hash algorithms will not suffer a security flaw from the quantum computers attack since they are not based on mathematical problems exploitable by the quantum algorithms developed so far.

It is worth noticing that one of the driving reasons for the quantum computer development is exactly the cryptographic application[4]

[2]High reliability and redundancy can make a system really expensive!

[3]A 4096 qbits [33] quantum computer is needed to break the RSA with the security parameters used today assuming the implementation of the current state of art Shor's algorithm.

[4]Former NSA contractor Edward Snowden reported that the U.S. National Security Agency (NSA) is running a $ 79.7 million research program (titled "Penetrating Hard Targets") to develop a quantum computer capable of breaking vulnerable encryption algorithm [58].

2.2 The R.S.A. algorithm

The most common crypto-system used in current cryptographic applications is RSA[5] [59]. The basic ingredient of the RSA is a mathematical function that is easy to perform but extremely hard to invert unless the user is in possession of a secret key, this concept is referred as *trapdoor one-way function.*

The RSA algorithm involves three major steps:

1. The **key generation**:

    - Choose two distinct prime numbers p and q. For security purposes they have to be approximately of same length. Their product should be an integer of 2048 bits therefore the prime numbers should be one thousand digits numbers.
    - Compute $n = pq$, the modulus used in both private and public operations and $\phi(n)$, the Euler's totient function. Euler's totient function acts on an integer input n counting the positive integers less than or equal to n that are relatively prime to n.
    - Choose an integer e between 1 and $\phi(n)$ coprime with $\phi(n)$ itself, usually e is chosen to be $2^{16} + 1$ for efficiency reasons: it is represented with 16 bits only and it is a sequence of two 1s and 14 zeros in binary representation.
    - Compute d, the multiplicative inverse of e modulo $\phi(n)$:

    $$d = e^{-1} \mod \phi(n) \tag{2.1}$$

    This task can be easily accomplished via the *extended Euclidean algorithm.*

    The *public key* comprises the public exponent e and the modulus n. The *private key* comprises the private modulus d and n as well because it is needed in the decryption algorithm. p, q and $\phi(n)$ must be kept secret because their knowledge allows to calculate the private exponent.

2. The **encryption**:

    - The encryption function takes as inputs a public key (e, n) and a message encoded as number m $0 \leq m \leq n$ and eventually sliced in packets and properly

[5]RSA is an acronym made from the first letters of the surnames of the three inventors: Ronald Rivest, Adi Shamir e Leonard Adleman.

padded. The ciphertext is simply the exponentiation of the message to the public exponent:

$$c = m^e \mod n \tag{2.2}$$

In a practical implementation each message needs to be randomly padded because the RSA alone suffers from a multiplication malleability, that means that it is possible to multiply two ciphertexts and still obtain a valid ciphertext. In certain situations this exploit is enough to break an implementation of the RSA [6].

3. The **decryption**:

   - the decryption function take as input the ciphertext c and the private key (d, n). The message m can be recovered from the ciphertext c by simply raising it to the private exponent:

$$m = c^d \mod n \tag{2.3}$$

     From m the original message can be recovered reversing the initial encoding.

The correctness of the decryption follows from the Fermat's little theorem. It states that if p is a prime number and it doesn't divide a number a then the following equation holds:

$$a^{p-1} = 1 \mod p \tag{2.4}$$

By similarity we can express the decryption as:

$$c^d = m^{de} = m^{de-1}m \mod n \tag{2.5}$$

where $de = 1 \mod \phi(n) = 1 \mod (p-1)(q-1)$ implies that $de-1 = k(q-1)(p-1)$ for $k \in \mathbb{N}$.

Substituting the last equality into the equation 2.5, remembering that $n = pq$ and using the Fermat's theorem we obtain:

$$c^d = m^{de} = m^{de-1} \cdot m = m^{k(q-1)(p-1)} \cdot m \mod pq = 1 \cdot mmodpq \tag{2.6}$$

This ends the small overview on the RSA algorithm (that together with the Elliptic Curves crypto-system) that is the cryptographic primitive most used in the practical implementations available on the market. The security of the system is based on the intractability of factoring big numbers even if the RSA problem is defined as the intractability of recovering the private exponent from the public one. Although this problem seems easier than the previous one there is no proof that this is the case. On the other hand, from the ability of factorizing big numbers follows the ability of solving the RSA problem.

It is clear that if an attacker factorizes a big number in a short time he could also easily recover $\phi(n)$ and find the private exponent using the extended Euclidean algorithm starting from the public exponent. As usual *short* means *polynomial time* in the number of digits of the security parameter. The major security parameter of the RSA is the public modulus, referred as key size. It is currently recommended that n be at least 2048 bits long.

2.3 Quantum Computer and Quantum Algorithms

I will now give a glimpse of what a quantum computer is and what the quantum algorithms are and how they work. This should explain why the research of alternatives is so important even if a fully operational quantum computer is not expected to become reality in the next two or three decades.

2.3.1 Quantum Computer

When we speak about quantum computers we should try not to compare them to the current computers: in general they are not better than the general purpose machines that we use every day. Using a standard application should be faster (and cheaper) on a standard machine for a lot of time from now. But quantum computers can exploit the rules of the quantum physics to accomplish few tasks in a more efficient way. One example is the famous *factorization of big numbers*. This is a problem considered very hard for a classic computer, and indeed it belongs the the NP complexity class, but becomes extremely easy if it is solved using a quantum computer and in the specific case the Shor's algorithm. Therefore the quantum computer is not better or faster in general but it is different. The main difference from a classic computer is that the logic of quantum computer follow the rules of the quantum physics, instead of the classic counterpart that uses classical physics.

The basic ingredients of the quantum computation are: the *superposition principle* and the *quantum entanglement* that follow directly from quantum physics [33, 16]. Another

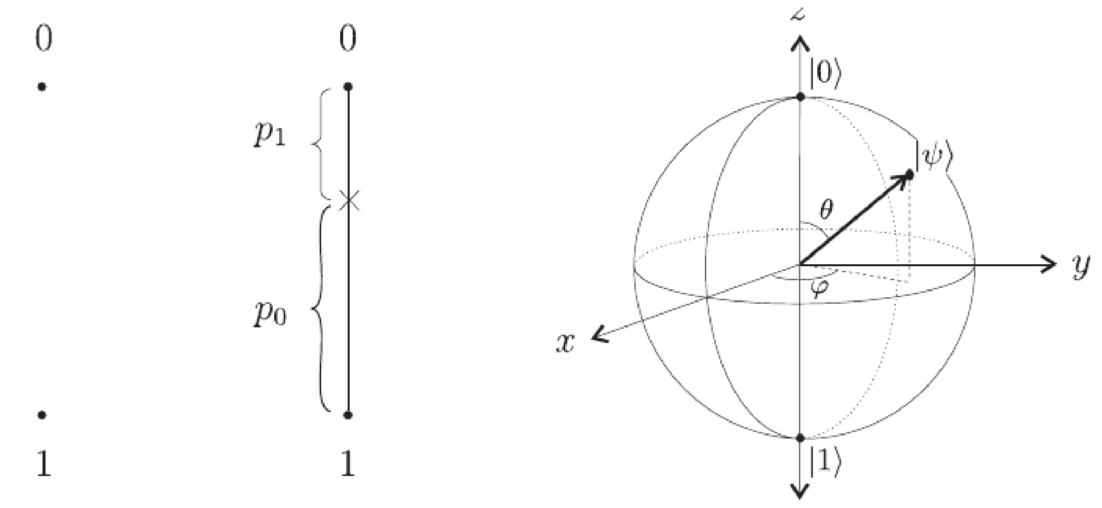


Figure 2.2: States of deterministic classical, probabilistic classical, and quantum bits.

important aspect of the quantum word is the probabilistic behavior exhibited by it. For example, if we consider the electron in an atom with two levels $|0\rangle, |1\rangle$, the function that describes its state in every instant is a superposition of the two states:

$$|\phi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{2.7}$$

where α and β are complex numbers and their square modulo is interpreted as the probability for the electron of being in the state $|0\rangle$ or $|1\rangle$ once it is measured and collapsed in one of those states. It is evident that if we wish to "code" this information we would need more than one bit, that is classically enough to code the state $|0\rangle$ and $|1\rangle$, but not enough to code the superposition of those two states. It is necessary to introduce the *qubit*, the unit of quantum information analogous to the classic bit, that brings with it in the computation also the *superposition principle*, the *quantum entanglement* and the *measurement principle*. The concept of the qubit can be explained using the Bloch sphere, see picture 2.2. In the picture are sketched in a pictorial way the deterministic classical bit, the probabilistic classical one and the quantum bit. From this comparison it is immediate to see that the qubit has an "additional dimension" usable for encoding information, even though this additional information are not directly accessible in a classic sense.

The extraordinary speed up of the quantum computer is related to the qubit. Quantum computer works on a larger amount of information at a time compared to the classical one: in a bit we can encode 0 or 1, instead on a qubit we can encode all the complex numbers. We can think that if we operate on a qubit we are operating on 0 and 1 at the same time, therefore the number of steps needed for a quantum computation, increasing the number of qubits used, have to be less than a classical one. Working with n qubits means working in a computation space of infinite size and dimension 2^n, in terms of algorithmic complexity: we are moving from a complexity asymptotically exponential to a polynomial one. This kind of parallelization is a bit tricky because operating on all the possible inputs doesn't produce all the possible outputs but only a superposition of them and in order to recover it we need to measure the output and it will collapse in only one state, according to the rules of the quantum mechanics. This is the price for such an outperforming speed up, the parallelization belongs to the private word of the qubits and we don't have access to all the possible results but only to an output that is at most of the same size of the input. Parallelization need be exploited in a proper way and the design of quantum algorithms need to take it into account.

A quantum algorithm can be seen like a set of classic algorithms that take place at the same time on a set of inputs, that usually are all the possible initial configurations of the considered problem. This is the first step in the algorithm building phase, the next step is to identify the computation steps for the state that is in superposition, with other words: what happen when the system is in a *coherent* state. The coherence is both the key for a quantum computation and one of the biggest obstacle. So far only the algorithms that exploit the parallelization and the interference can solve current hard problems in a reasonable amount of time.

So far it was possible to keep few qubits in a coherent state: too less for a practical usage of a quantum computer. It is extremely easy to loose the quantum coherence, as we know from the quantum physics any kind of measure (accidental or wanted) can compromise it so the practical realization of a quantum computer is a big challenge. As rule of thumb we can think that the coherence time has to be much larger than the time needed to perform any kind of logic operation: $t_{coh} \gg t_{gate}$. There are few criteria that a quantum computer implementation has to fulfill, these are known as the Di Vincenzo's criteria [16].

For sake of completeness I will briefly list some of the practical implementations tried so far:

- Trapped Ions;
- Atoms in optical lattice;
- Qubits in superconductors;
- Electron in quantum dots;

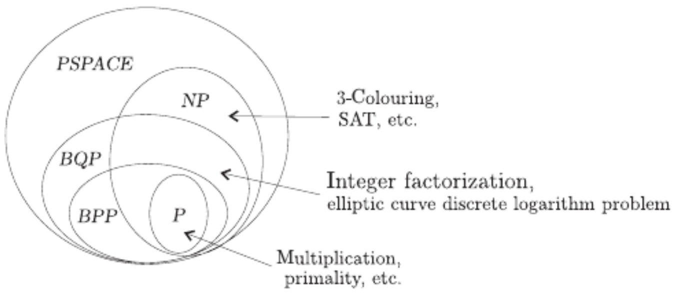

PSPACE
NP
BQP
BPP
P
3-Colouring, SAT, etc.
Integer factorization, elliptic curve discrete logarithm problem
Multiplication, primality, etc.

Figure 2.3: Complexity classes introduced with the introduction of the quantum computation.

- Magnetic Resonance;
- Photons.

2.3.2 Shor's algorithm

The most important quantum algorithm for a cryptographic perspective is the *Shor's* one [19]. It allows to factorize a number in polynomial time with a certain probability. This algorithm redefine the complexity classes and make necessary to introduce the **BQP** class: Bounded-error Quantum Polynomial-time, that contains all the problems solvable with a quantum algorithm with an error equal or less than one third. One of this problem is of course the factorization that belongs to the **BQP** class, see picture 2.3.

The algorithm is made up of a classic part, the mathematical ingredients, and a quantum part that allows the sped up. Now let us look more closely at the quantum part of the algorithm. Some of the key quantum operations can be thought of as looking for certain kinds of patterns in a superposition of states. Because of this, it is helpful to think of the algorithm as having two stages. In the first stage, the n classical bits of the input are "unpacked" into an exponentially large superposition, which is expressly set up so as to have an underlying pattern or regularity that, if detected, would solve the task at hand. The second stage then consists of a suitable set of quantum operations, followed by a measurement, which reveals the hidden pattern.

The algorithm to factor a large integer N can be viewed as a sequence of reductions:

- FACTORING is reduced to finding a non-trivial square root of 1 modulo N.
- Finding such a root is reduced to computing the order of a random integer modulo N.
- The order of an integer is precisely the period of a particular periodic superposition.

- Finally, periods of superpositions can be found by the quantum FFT.

Starting from the bottom of the list: we can see the *Quantum Fourier Transform* (QFT) [33] as a variant of the classical Fourier transform. The QFT maps a quantum state $|x\rangle$ to quantum state $|y\rangle$ through the relation:

$$QFT_m : |x\rangle \longmapsto \frac{1}{\sqrt{m}} \sum_{y=0}^{m-1} e^{2\pi i \frac{x}{m} y} |y\rangle \tag{2.8}$$

where m, in practical implementation, has to be a power of 2. The main difference between the classical FFT and the quantum version is that the classic algorithm outputs a vector of complex numbers of the same size of the input vector. In contrast the QFT only prepares a superposition of these numbers, since the probability distributions of the components of this superposition belong to the private world. When we perform a measurement we only get an output y with a certain probability. y is a single number and can be seen as the component of the input vector with the largest weight.

Quantum Fourier sampling is basically a quick way of getting a very rough idea about the output of the classical FFT, just detecting one of the larger components of the answer vector. In fact, we don't even see the value of that component. We only see its index. How can we use such information?

Suppose that the input to the QFT, $|\alpha\rangle = (\alpha_0, \alpha_1, \dots, \alpha_{M-1})$, is such that $\alpha_i = \alpha_j$ whenever $i = j \, mod \, k$ where k divides M. That is the array α consists of M/k repetitions of a sequence of length k. Moreover, suppose that exactly one of the k number of list is non-zero, let's say α_j. Then we can say that $|\alpha\rangle$ is *periodic with period k and offset j.*

It turns out that if the input vector is periodic, we can use quantum Fourier sampling to compute its period! This is based on the following fact (the proof is skipped): suppose the input to quantum Fourier sampling is periodic with period k, for some k that divides M. Then the output will be a multiple of Mk, and it is equally likely to be any of the k multiples of M/k. Repeating the sampling a few times (repeatedly preparing the periodic superposition and doing Fourier sampling), and then taking the greatest common divisor of all the indices returned, we will get, with very high probability, the number M/k and from it the period k of the input!

We have seen how the quantum Fourier transform can be used to find the period of a periodic superposition. We can now see, by a sequence of simple reductions, how factoring can be recast as a period-finding problem.

Fix an integer N. A nontrivial square root of 1 modulo N is any integer $x \neq \pm 1 \, mod \, N$ such that $x^2 = 1 \, mod \, N$. If we can find a nontrivial square root of $1 mod N$, then it is easy to decompose N into a product of two nontrivial factors (and repeating the process would factor N): If x is a nontrivial square root of 1 modulo N, then $gcd(x+1;N)$ is a nontrivial factor of N.

To complete the connection with periodicity, one further concept is needed. Define the order of x modulo N to be the smallest positive integer r such that $x^r = 1\, mod\, N$. Computing the order of a random number $x\, mod\, N$ is closely related to the problem of finding nontrivial square roots, and thereby to factoring: let N be an odd composite, with at least two distinct prime factors, and let x be chosen uniformly at random between 0 and $N-1$. If $gcd(x;N)=1$, then with probability at least 1/2, the order r of $x\, mod\, N$is even, and moreover $x^{r/2}$ is a nontrivial square root of $1\, mod\, N$. In which case $gcd(x^{r/2}+1;N)$ is a factor of N. Hence we have reduced FACTORING to the problem of ORDER FINDING. The advantage of this latter problem is that it has a natural periodic function associated with it: $f(a) = x^a modN$ with period r. The last step is to figure out how to use the function f to set up a periodic superposition with period r; whereupon we can use quantum Fourier sampling to find r:

- Start with two quantum registers, both initially 0.
- Compute the quantum Fourier transform of the first register modulo M, to get a superposition over all numbers between 0 and $M-1$: $\frac{1}{\sqrt{M}}\sum_{a=0}^{M-1}|a,0\rangle$. This is because the initial superposition can be thought of as periodic with period M, so the transform is periodic with period 1.
- Compute the function $f(a) = x^a modN$. The quantum circuit for doing this regards the contents of the first register a as the input to f, and the second register (which is initially 0) as the answer register. After applying this quantum circuit, the state of the two registers is: $\frac{1}{\sqrt{M}}\sum_{a=0}^{M-1}|a,f(a)\rangle$
- Measure the second register. This gives a periodic superposition on the first register, with period r, the period of f. Since f is a periodic function with period r, for every rth value in the first register, the contents of the second register are the same. The measurement of the second register therefore yields $f(k)$ for some random k between 0 and $r-1$. What is the state of the first register after this measurement? Due to rules of partial measurement (i.e. the quantum entanglement) the first register is now in a superposition of only those values a that are compatible with the outcome of the measurement on the second register. But these values of a are exactly $k; k+$

$r; k + 2r; \ldots; k + M - r$. So the resulting state of the first register is a periodic superposition $|\alpha\rangle$ with period r, which is exactly the order of x that we wish to find!

This ends the short overview on the Shor's algorithm, the most important threat, coming from the quantum world, to the today's crypto-systems based on the hardness of the factoring problem.

2.3.3 Grover's algorithm

The last quantum algorithm that is relevant for cryptographic purposes is the Grover's one [4, 25]. Its purpose is to speed up the searching process in an unsorted database by a quadratic factor in the complexity. That means that it is not so devastating like the previous one but it needs to be taken into account when we choose the key size thus the key space that an attacker would need to search in order to find the right key. Both symmetric and asymmetric crypto-system are vulnerable to this algorithm. Let's now give a not so deep look to this algorithm.

The algorithm essentially define an operator that has as sub-routine an oracle O that is able to tell if a candidate is a solution or not for the given problem. Invoking O, together with other operations, on the superposition of all the possible candidates determines an increasing of the probability amplitude of the solution among all the possible candidates in the superposition. The new superposition will have the highest amplitude on the component that is the solution and if we perform a measurement on the superposition we will be likely to get this component as result. This technique can be seen as a series of rotation applied to the initial vector (i.e. the superposition of all the possible candidates) that bring the initial vector closes to the target vector.

Speaking about the cryptographic application: if an attacker wish to find a secret key he can only try a brute force attack, that means if he wish to find a key of length n he will have to search a space of dimensions 2^n and in the worst scenario he will have to try all the 2^n keys to find the right one. If this attacker was in possession of a quantum computer the problem would still be difficult with an asymptotical exponential complexity but he would have a boost of a quadratic factor, it is like he would have to search in a space of size $2^{n/2}$. To put it in numbers: a classical attacker that is able to perform 100 millions test per second that wish to retrieve a 100bits key (with a key space size of approximatively 10^{30}) will have to make something like 10^{29} (that is the half of the search space) tests. At 100 millions test per second he will need 10^{21} seconds, a time close to the age of the Universe. The same attacker with the same capability of tests per second but with a quantum computer will have to search a space of the size 10^{15} and he will be able to do it in only 10^7 seconds, roughly four months [16]. It is evident that to neutralize this advantage is enough to increase the size of the keys, an operation not so difficult with the today's hardware and communication speed.

Chapter 3

Mainstream alternatives to the current PKC

In this first part of my thesis I will briefly explain the working principles of the PKC's mainstream alternatives. I did not focus my efforts on the cryptanalysis, which is one of the main aspect in the design of a PKC, rather I focused my attention on the basic mathematical operations performed during an encryption/decryption cycle. Most of this proposal are not yet suitable for the digital signature neither [42], this is why I only took in account the encryption/decryption capabilities.

I will also build a framework in which categorize these algorithms trying to give enough and valid reasons to motivate why I chose to further investigate one lattice-based PKC. The families of algorithms, supposed to resist to a quantum attack are not based on the *Hidden Subgroup Problem.* In its simplest form *Hidden Subgroup Problem* can be defined as: given a periodic function f on $\mathbb{Z}$ to find its period, i.e. to find the hidden subgroup $l\mathbb{Z}$ of $\mathbb{Z}$ of smallest index for which f is constant on the cosets $a + l\mathbb{Z}$. As already showed in the introduction the factorization problem belongs to this class of problems. The families of crypto-systems under investigation are:

- Code-based;
- Lattice-based;
- MPKC;

The *Code-based crypto-systems* use algorithmic primitives based on the long standing problem of the errors correction that usually take place in a communication. The error correction problem is believed to be hard in general.

The *Lattice-based crypto-systems* use primitives based on different type of lattice problems. A lattice is a periodic collection of points defined by a basis. Examples of lattice problems are: the shortest vector problem and the learning with errors problem. The hardness of these problems is in general based on the intractability of high dimensions lattices problems.

The *MPKC* is a PKC that uses as trapdoor a set of polynomials, usually quadratic ones, with a large number of variables over a finite field. The hardness of this type of PKCs is based on the assumption that to solve a set of random multivariate polynomials is NP-hard.

In the next sections I will give some hints on different crypto-systems that belong to these families of "quantum resistant problems". Finally I will build a framework with different parameters mostly based on a "practical points of view". Inside this framework I will categorize these algorithms choosing one for further investigation/implementation and enhancements.

3.1 Code-based crypto-systems

Code-based crypto-systems are crypto-systems in which the algorithmic primitives uses and error correcting code. Let $\mathcal{C}$ denote an $[n,k]$ linear code over a finite field $\mathbb{F}_q$. The primitive may consist in adding an error to a word of $\mathcal{C}$ or in computing a syndrome relatively to a parity check matrix of $\mathcal{C}$. The syndrome is defined as the scalar product between the an error vector and the transpose of a parity check matrix. And a parity check matrix of $\mathcal{C}$ is defined as the generator matrix of $\mathcal{C}^{\perp}$ where the orthogonal of $\mathcal{C}$ is defined for the usual scalar product over a finite field $\mathbb{F}_q$.

The first of these system is a public key encryption scheme named after his inventor: McElice. There is also a variant of the McElice version and it is referred as the Niederreiter variant. The security of these crypto-systems is equivalent, this mean that an attacker that is able to break one is able to break the other and vice versa [9].

3.1.1 McElice crypto-system

The McElice crypto-system uses as private key a random binary irreducible Goppa code $\mathcal{C}$ and the public key is a random generator matrix[1] of a randomly permuted version of that code [41, 32].

[1]A generator matrix G for $\mathcal{C}$ is a matrix over $\mathbb{F}_q$ such that $\mathcal{C} = \langle G \rangle$, where $\langle G \rangle$ denotes the vector space spanned by the rows of G.

Before proceed further it is worth to define the irreducible binary Goppa code.

A binary Goppa code $\mathcal{C}$ over $\mathbb{F}_{2^m}$ is defined by a vector $\mathbf{a} \in \mathbb{F}_{2^m}^n$, where $a_i \neq a_j$ and a Goppa polynomial $g(X) = \sum_{i=0}^{t} g_i X^i \in \mathbb{F}_{2^m}[X]$.

$\mathcal{C}$ is the set of all $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{F}_2^n$ such that the identity:

$$S_c(X) = -\sum_{i=0}^{n-1} \frac{\mathbf{c}_i}{g(a_i)} \frac{g(X) - g(a_i)}{X - a_i} \pmod{g(X)} \tag{3.1}$$

holds in the polynomial ring $\mathbb{F}_{2^m}[X]$.

The Goppa polynomial used for cryptographic purposes has to be irreducible, this means, by definition, that the Goppa code has a minimum distance of $2t + 1$ and is capable to correct up to t errors. The distance or minimum distance of a linear code is defined as the minimum weight of its non zero codewords. Roughly speaking, the weight is the number of ones in a binary codeword. Ultimately, using a physical interpretation, the minimum distance notion is related to the lightest codeword,i.e. closer to a zero codeword, that is resolvable by particular code.

Encryption and Decryption Overview

The system parameters are: $n, k, t \in \mathbb{N}$, where t is weight of the error vector and n, k are the dimensions of the generator matrix G of the code $\mathcal{C}$ i.e. $G \in \mathbb{F}_q^{k \times m}$. The keys generation can take place once we set these parameters. First we need to compute the three matrices:

- $G : k \times n$ generator matrix of a code $\mathcal{C}$ over $\mathbb{F}_q$ of dimension k and minimum distance $d \geq 2t + 1$;
- $S : k \times k$ random binary non-singular matrix;
- $P : n \times n$ random permutation matrix;

Then we can compute the $k \times n$ matrix $G^{pub} = SGP$ from the previous ones. Briefly the keys and the encryption/decryption procedures are:

- the *public key* is (G^{pub}, t);
- the *private key* is $(S, \mathcal{D}_{\mathcal{C}}, P)$, where $\mathcal{D}_{\mathcal{C}}$ is an efficient decoding algorithm[2] for $\mathcal{C}$;

[2]A decoder for $\mathcal{C}$ is mapping $\mathcal{D}_{\mathcal{C}} : \mathbb{F}_q^n \to \mathcal{C}$. It is t-error correcting if for all $\mathbf{e} \in \mathbb{F}_q^n$ and all $\mathbf{x} \in C$ $wt(\mathbf{e}) \leq t \Rightarrow \mathcal{D}_{\mathcal{C}}(\mathbf{x} + \mathbf{e}) = \mathbf{x}$. Furthermore, for any linear code there exist a t-error correcting decoder iff $t < d/2$.

- to perform the *encryption* of a plaintext $\mathbf{m} \in \mathbb{F}^k$ a random vector $\mathbf{z} \in \mathbb{F}^n$ of weight t as to be chosen in order to compute the ciphertext $\mathbf{c}$ as follows:

$$\mathbf{c} = \mathbf{m}\mathbf{G}^{pub} \oplus \mathbf{z}^3; \tag{3.2}$$

- the *decryption* process is made first applying $\mathbf{P}^{-1}$ to the ciphertext obtaining:

$$\mathbf{c}\mathbf{P}^{-1} = (\mathbf{m}\mathbf{S})\mathbf{G} \oplus \mathbf{z}\mathbf{P}^{-1} \tag{3.3}$$

  After this we apply the decoding algorithm $\mathcal{D}_{\mathcal{C}}$ to $\mathbf{c}\mathbf{P}^{-1}$. Since $\mathbf{c}\mathbf{P}^{-1}$ has a Hamming distance of t we will recover $\mathbf{m}\mathbf{S}\mathbf{G} = \mathcal{D}_{\mathcal{C}}(\mathbf{c}\mathbf{P}^{-1})$. The Hamming distance between two codewords, of equal length, is the number of positions at which the corresponding symbols are different. Assuming G^{pub} is invertible, with the knowledge of the private key we will be able to recover $\mathbf{m}$ from $\mathbf{m}\mathbf{S}\mathbf{G}$.

3.1.2 Niederreiter variant

The difference between this variant and the McElice one is that in the Niederreiter variant the message is encoded into an error vector by a function $\phi_{n,t}$:

$$\phi_{n,t} : \{0,1\}^l \rightarrow \mathcal{W}_{n,t} \tag{3.4}$$

where $\mathcal{W}_{n,t} = \{\mathbf{e} \in \mathbb{F}_2^n \mid wt(\mathbf{e})\}$ and $l = \lfloor \log_2 |\mathcal{W}_{n,t}| \rfloor$. The Niederreieter variant in combination with the Goppa codes is equivalent (in security) to the McElice proposal and with the right parameter choice it is still unbroken. In the original proposal instead of the Goppa codes were used the GRS codes that was broken in 1992 because the GRS code has a well defined algebraical structure that can be exploited.

The major advantage of this variant is that the public key has a smaller size because it is sufficient to store the redundant part of the matrix H^{pub} (see below). The disadvantage is that the mapping $\phi_{n,t}$ slows encryption and decryption procedures.

Encryption and Decryption Overview

The system parameters are: $n, k, t \in \mathbb{N}$, with the same meaning of the parameters for the McElice crypto-system. As for the McElice variant we need a set of three matrices, defined by the same parameters, to generate the key.

[3]Where the $\oplus$ is the XOR operator.

- $H : (n-k) \times n$ parity check matrix of a code $\mathcal{C}$ which can correct up to t errors;
- $M : (n-k) \times (n-k)$ random binary non-singular matrix;
- $P : n \times n$ random permutation matrix;

Then compute the systematic $(n-k) \times n$ matrix $H^{pub} = MHP$, whose columns span the column space of HP, i.e. $H^{pub}_{1,\dots,n-k}$ and can be written as I_{n-k}[4].

Again we can define the keys and the procedures:

- the *public key* will be (H^{pub}, t);
- the *private key* will be: $(M, \mathcal{D}_{\mathcal{C}}, P)$, where $\mathcal{D}_{\mathcal{C}}$ is an efficient syndrome decoding algorithm[5] for $\mathcal{C}$;
- a message is represented as a vector $\mathbf{e} \in 0, 1^n$ of weight t: the plaintext. To perform the *encryption* compute the syndrome $\mathbf{s}$:

$$\mathbf{s} = H^{pub}\mathbf{e}^T \tag{3.5}$$

- to *decrypt* a ciphertext $\mathbf{s}$ calculate

$$M^{-1}\mathbf{s} = HP\mathbf{e}^T \tag{3.6}$$

  first, and then apply the syndrome decoding algorithm $\mathcal{D}_{\mathcal{C}}$ to $M^{-1}\mathbf{s}$ in order to recover $P\mathbf{e}^T$. The plaintext can then easily obtained applying P^{-1} to $\mathcal{D}_{\mathcal{C}}(M^{-1}\mathbf{s}) = P\mathbf{e}^T$ and retrieve $\mathbf{e}^T$. The message is then recovered from the plaintext applying to it the inverse of the map $\phi_{n,t}$ previously defined.

3.2 Lattice-based crypto-systems

A lattice is a set of points in n-dimensional space with a periodic structure. More formally, given n-linearly independent vectors $(\mathbf{b}_1, \dots, \mathbf{b}_n) \in \mathbb{R}^n$, the lattice generated by them is the set of vectors:

$$\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_n) = \sum_{i=1}^{n} a_i\mathbf{b}_i : a_i \in \mathbb{Z} \tag{3.7}$$

[4]In this way is possible to reduce the public key dimensions storing only

[5]A syndrome decoding algorithm takes as input a syndrome, not a codeword as was for the McElice crypto-system.

The vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ are known as a basis of the lattice. The number of bases of a lattice is of course infinite, some of them are "good basis" for cryptographic purposes and some other are "bad basis". The good ones are almost orthogonal basis $\mathbf{B}$, this means that the lattice problems in this particular basis are very simple, this is can be intended to be the private key of a PKC. The bad basis are ones with vectors far away from the orthogonal form: i.e. $\mathbf{B'} = \mathbf{UB}$, where $\mathbf{U}$ is an unimodular matrix and $\mathbf{B'}$ can be viewed as public keys.

It is also worth mentioning "q-ary Lattices": they are of particular interest in cryptography [10]. In this type of lattice, the membership of a vector $\mathbf{x} \in \mathcal{L}$ is determined by the relation $\mathbf{x} \pmod q$. Such lattices are in one to one correspondence with the linear codes in $\mathbb{Z}_q^n$.

Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ for some integers q, m, n, we can define two m-dimensional q-ary lattices:

- $\Lambda_q(\mathbf{A}) = \mathbf{y} \in \mathbb{Z}^m : \mathbf{y} = \mathbf{A}^T\mathbf{s} \pmod q$ for some $\mathbf{s} \in \mathbb{Z}^n$
- $\Lambda_q^\perp = \mathbf{y} \in \mathbb{Z}^m : \mathbf{Ay} = \mathbf{0} \pmod q$

The first q-ary lattice is generated by the rows of $\mathbf{A}$; the second contains all vectors that are orthogonal modulo q to the rows of $\mathbf{A}$.

The lattice offer a very rich variety of problems upon which is possible to build crypto-system primitives. The most known computational problems (but not supported by a theoretical prof of security are) [10, 51]

- Shortest Vector Problem (SVP): given a lattice basis $\mathbf{B}$ find the shortest nonzero vector in the lattice generated by this basis $\mathcal{L}(\mathbf{B})$.
- Closest Vector Problem (CVP): give a lattice basis $\mathbf{B}$ and a target vector $\mathbf{t}$ (not necessarily in the lattice), find the lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ closest to $\mathbf{t}$.
- Shortest Independent Vector Problem (SIVP): given a lattice basis $\mathbf{B} \in \mathbb{Z}^{n\times n}$, find n linearly independent lattice vectors $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_n]$ (where $\mathbf{s}_i \in \mathcal{L}(\mathbf{B})\text{for all}i$) minimizing the quantity $\|\mathbf{S}\| = \max_i \|\mathbf{s}_i\|$.

In lattice-based cryptography, one typically considers the approximation variant of these problems. For instance, in SVP_γ the goal is to find a vector whose norm is at most γ times that of the shortest nonzero vector.

Let's give a short overview on PKC based on the SVP problem focusing on the encryption and decryption processes. Furthermore there is another PKC defined in the lattice

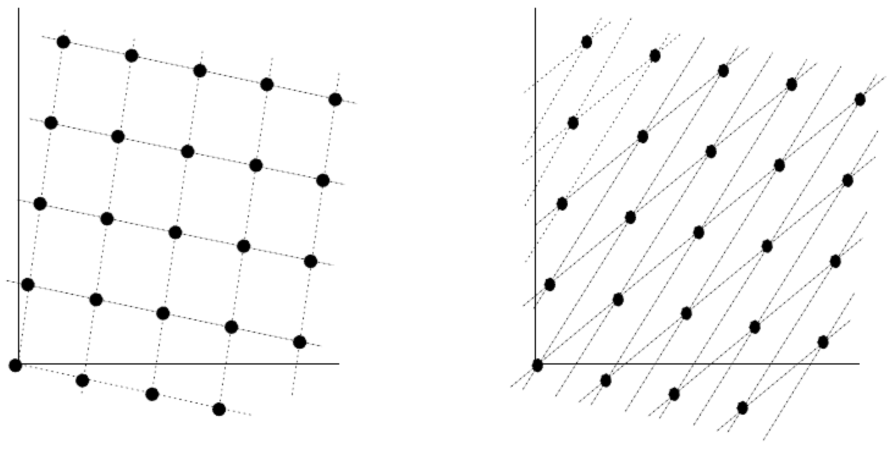


Figure 3.1: Two examples of the same lattice in R^2, described using two different bases, the first one is a good basis: i.e. the private key and vice versa.

framework but based on another interesting family of algorithmic problems: the Learning with Error Problem (LWE).

3.2.1 The GGH/HNF Public Key crypto-system

The GGH PKC is one of the first attempt to built a PKC based on lattice problems and it is an analogue to the McElice crypto-system. The security of this crypto-system relies on the assumption that without knowledge of a special basis, solving the instance of the CVP is computationally hard.

The system, although no asymptotically good attack is known [49], is impractical because general lattice bases require $\Omega(n^2)$ storage but it is a good “toy” to understand the logic behind the lattice based crypto-systems and a starting point from which is possible to build more efficient systems.

Encryption and Decryption Overview

- The *private key* is a “good”[6] lattice basis $\mathbf{B}$.
- The *public key* is a “bad” basis $\mathbf{H}$ for the same lattice, see picture 3.1. An interesting proposal is to use lower triangular form for the public basis (Hermite Normal Form, HNF) [63, 44, 43], since they are efficiently calculable.
- The *encryption* process consists of adding a short noise vector $\mathbf{r}$ (somehow encoding the message to be encrypted) to a properly chosen lattice point $\mathbf{v}$. A common procedure, also to other PKC, is to select the vector $\mathbf{v}$ such that all coordinates

[6]Almost all the basis vector are orthogonal and short.

of $(\mathbf{r}+\mathbf{v})$ are reduced modulo the corresponding element along the diagonal of the public basis $\mathbf{H}$. The resulting vector is denoted as $\mathbf{r} \pmod{\mathbf{H}}$, i.e. $(r+v)_i \pmod{h_{i,i}}$ is the i-th coordinate of $\mathbf{r} \pmod{\mathbf{H}}$.

- The *decryption* process corresponds to find the lattice point $\mathbf{v}$ closest to the chiper-text $\mathbf{c} = \mathbf{r} \pmod{\mathbf{H}} = (\mathbf{v}+\mathbf{r})$ and the associated error vector $\mathbf{r}$. This can done using the private basis $\mathbf{B}$ trough the Babai's rounding procedure [3]

$$\mathbf{v} = \mathbf{B}[\mathbf{B}^{-1}(\mathbf{v}+\mathbf{r})] \tag{3.8}$$

The security relies on the assumption that without knowing the private basis this problem is hard as a CVP instance.

3.2.2 NTRU crypto-system

The NTRU can be described as an instance of the general GGH crypto-system [26, 29, 14]. Furthermore this particular crypto-system has a well known dual construction, i.e. as happens often in these families of quantum resistant crypto-systems they can be described either in the matrices ring or in the polynomials ring with an overlap in the notation.

Before we can describe the crypto-system we need to define a linear transformation $\mathbf{T}$ in order to define the circulant matrix [24] that will be the basis for the lattice that we will use in this crypto-system.

$$\mathbf{T} = \left[\begin{array}{ccc|c} 0 & \dots & 0 & 1 \\ \hline \ddots & & & 0 \\ & \mathbf{I} & & \vdots \\ & & \ddots & 0 \end{array}\right] \tag{3.9}$$

is the permutation matrix $(n \times n)$ that rotates the coordinates of a vector cyclically. We can now define $\mathbf{T}^*\mathbf{v} = [\mathbf{v}, \mathbf{T}\mathbf{v}, \dots, \mathbf{T}^{n-1}\mathbf{v}]$ as the circulant matrix of the vector $\mathbf{v} \in \mathbb{Z}^n$.

The NTRU lattices, named *convolutional modular lattice*, are lattices in even dimension $2n$ that satisfy two properties:

1. they are closed under the linear transformation that maps the vector $(\mathbf{x}, \mathbf{y})$ to $(\mathbf{T}\mathbf{x}, \mathbf{T}\mathbf{y})$;
2. they are $q - ary$ lattices, hence the membership of a pair $(\mathbf{x}, \mathbf{y})$ in the lattice is of the type: $(\mathbf{x}, \mathbf{y}) \pmod{q}$.

The system parameter are a prime dimension n, an integer modulus q, a small integer p and an integer weight bound d_f. We can now describe the encryption and decryption procedure of the NTRU crypto-system.

Encryption and Decryption Overview

- The *private key* is a short vector $(\mathbf{f}, \mathbf{g}) \in \mathbb{Z}^{2n}$. The lattice associated to to a private key $(\mathbf{f}, \mathbf{g})$ is by definition: $\Lambda_q((\mathbf{T}^*\mathbf{f}, \mathbf{T}^*\mathbf{g})^T)$, which can be seen as the smallest convolutional modular lattice containing $(\mathbf{f}, \mathbf{g})$. The secret vector $(\mathbf{f}, \mathbf{g})$ are subject to the following technical restrictions:

  - the matrix $[\mathbf{T}^*\mathbf{f}]$ has to be invertible modulo q;
  - $\mathbf{f} \in \mathbf{e}_1 + \{p, 0, -p\}^n$ and $\mathbf{g} \in \{p, 0, -0\}^n$ are randomly chosen polynomials such that $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ have exactly $(d_f + 1)$ positive entries and d_f negative ones. The remaining entries will be zero. Technical aside: the NTRU crypto-system has a dual construction, one is based on a vector formulation and the other one on polynomials: $\mathbf{f}$ is either a vector $\in \mathbb{Z}_q^n$ or a polynomial in the ring $Z[X]/(x^n - 1)$, $\mathbf{f}$ is a vector of n coefficients of a polynomial of degree $n-1$ or just a vector.

  The bounds on the number of zero entry is motivated by efficiency reasons. More important are the requirements on the invertibility of $[\mathbf{T}^*\mathbf{f}]$ modulo q, and the restriction of $\mathbf{f} - \mathbf{e}_1$ and $\mathbf{g}$ to the set $\{p, 0, -p\}^n$, which are used in the public key computation, encryption and decryption operations. Under these restriction $[\mathbf{T}^*\mathbf{f}] \equiv \mathbf{I} \pmod{p}$ and $[\mathbf{T}^*\mathbf{g}] \equiv \mathbf{O}^7 \pmod{p}$.

- The *public key* correspond to the Hermite Normal Form basis of the convolutional modular lattice $\Lambda_q((\mathbf{T}^*\mathbf{f}, \mathbf{T}^*\mathbf{g})^T)$ defined by the private key. Due to the structural properties of convolutional modular lattices, and the restrictions on the choice of $\mathbf{f}$, the HNF public basis has a nice form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{T}^*\mathbf{h} & q\mathbf{I} \end{bmatrix} \tag{3.10}$$

  and can be compactly represented just by the vector $\mathbf{h} \in \mathbb{Z}_q^n$, where $\mathbf{h} = [\mathbf{T}^*\mathbf{f}]^{-1}\mathbf{g} \pmod{q}$.

[7] $\mathbf{O}$ denotes all the zero matrix

- To *encrypt* a message in a lattice first we need to encode it as a vector $\mathbf{m} \in \{1, 0, -1\}^n$ with exactly $(d_f + 1)$ positive entries and d_f negative ones. The vector $\mathbf{m}$ is concatenated with a randomly chosen vector $\mathbf{r} \in \{1, 0, -1\}^n$ also with exactly $(d_f + 1)$ positive entries and d_f negative ones, to obtain a short error vector $(-\mathbf{r}, \mathbf{m}) \in \{1, 0, -1\}^{2n}$.The restriction on the number of nonzero entries is used to bound the probability of decryption errors. Reducing the error vector $(-\mathbf{r}, \mathbf{m})$ modulo[8] the public basis $\mathbf{H}$ yields:

$$\begin{bmatrix} -\mathbf{r} \\ \mathbf{m} \end{bmatrix} \pmod{\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{T}^*\mathbf{h} & q\mathbf{I} \end{bmatrix}} = \begin{bmatrix} \mathbf{0} \\ (\mathbf{m} + [\mathbf{T}^*\mathbf{h}]\mathbf{r}) \pmod q \end{bmatrix} \tag{3.11}$$

  Since the first n coordinates of this vector are always 0, they can be omitted, leaving only the n-dimensional vector $\mathbf{c} = \mathbf{m} + [\mathbf{T}^*\mathbf{h}]\mathbf{r} \pmod q$ as the ciphertext.

- The *decryption* process of $\mathbf{c}$ is made by multiplying it by the secret matrix $[\mathbf{T}^*\mathbf{f}] \pmod q$, yielding:

$$\begin{aligned}[\mathbf{T}^*\mathbf{f}]\mathbf{c} \pmod q &= [\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{f}][\mathbf{T}^*\mathbf{h}]\mathbf{r} \pmod q \\ &= [\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r} \pmod q, \end{aligned} \tag{3.12}$$

  where we have used the identity $[\mathbf{T}^*\mathbf{f}][\mathbf{T}^*\mathbf{h}] = [\mathbf{T}^*([\mathbf{T}^*\mathbf{f}]\mathbf{h})]$ valid for any vectors $\mathbf{f}$ and $\mathbf{h}$. The decryption procedure relies on the fact that the coordinates of the vector $[\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r}$ are all bounded by $q/2$ in absolute value, so the decrypter can recover the exact value of it over the integers (i.e., without reduction modulo q.) The decryption process is completed by reducing $[\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r}$ modulo p, to obtain $[\mathbf{T}^*\mathbf{f}]\mathbf{m} + [\mathbf{T}^*\mathbf{g}]\mathbf{r} \pmod p = \mathbf{Im} + \mathbf{Or} = \mathbf{m}$.

No proof of security supporting the NTRU is known, and confidence in the security of the scheme is gained primarily from the best currently known attacks [30, 31, 49].

3.2.3 LWE-Based crypto-system

This crypto-system was shown to be secure (under chosen plaintext attacks [5]) based on the conjectured hardness of the learning with errors problem (LWE) [51, 46]. This problem is parameterized by integers n, m, q and a probability distribution χ on $\mathbb{Z}_q$, typically taken to be a “rounded” normal distribution.

The input is a pair $(\mathbf{A}, \mathbf{v})$ where $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ is chosen uniformly, and $\mathbf{v}$ is either chosen uniformly from Z_q^m or chosen to be $\mathbf{As} + \mathbf{e}$ for a uniformly chosen $\mathbf{s} \in \mathbb{Z}_q^n$ and a vector

[8]The reduction modulo a matrix has the same meaning of the GGH crypto-system previously shown.

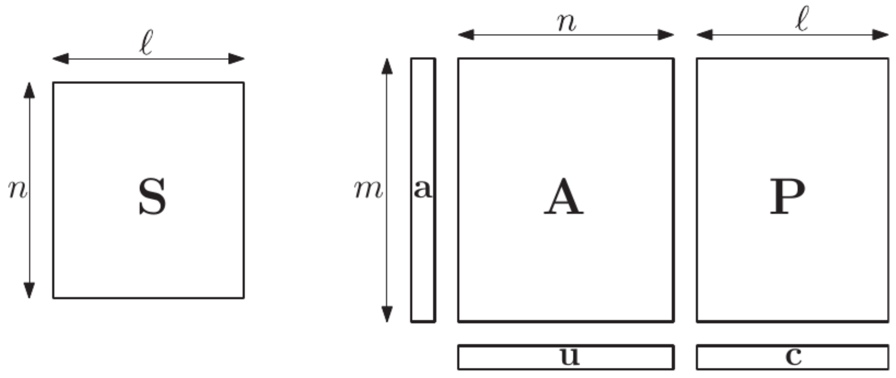


Figure 3.2: The objects defining the LWE crypto-system.

$\mathbf{e} \in \mathbb{Z}_q^m$ chosen according to χ^m. The goal is to distinguish with some non-negligible probability between these two cases.

This problem can be equivalently described as a bounded distance decoding problem in $q-ary$ lattices: given a uniform $\mathbf{A} \in \mathbf{Z}_q^{m\times n}$ and a vector $\mathbf{v} \in \mathbb{Z}_q^m$ we need to distinguish between the case that $\mathbf{v}$ is chosen uniformly from $\mathbb{Z}_q^m$ and the case in which $\mathbf{v}$ is chosen by perturbing each coordinate of a random point in $\Lambda_q(\mathbf{A}^T)$ using χ.

The LWE problem is believed to be very hard (for reasonable choices of parameters), with the best known algorithms running in exponential time in the lattice dimentions n. Several other facts lend credence to the conjectured hardness of LWE. First, the LWE problem can be seen as an extension of a well-known problem in learning theory [15], known as the learning parity with noise problem, which in itself is believed to be very hard. Second, LWE is closely related to decoding problems in coding theory which are also believed to be very hard.

The crypto-system is mainly defined by the security parameter n that is the lattice dimension. Furthermore we need to define a function f, known as encoder, to be the function that maps the message space $\mathbb{Z}_t^l \to \mathbb{Z}_q^l$ by multiplying each coordinate by q/t and rounding to the nearest integer.

Encryption and Decryption Overview

- The private key is a uniformly chosen at random matrix $\mathbf{S} \in \mathbb{Z}_q^{n\times l}$.
- The public key is built starting with a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and a matrix $\mathbf{E} \in \mathbb{Z}_q^{m\times l}$, which entries are chosen according to the distribution ϕ_α. The public key is defined as: $(\mathbf{A}, \mathbf{P} = \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{m\times l}$.
- The encryption is straightforward: given an element of the message space $\mathbf{v} \in \mathbb{Z}_t^l$, a

public key $(\mathbf{A},\mathbf{P})$ and a uniformly random chosen vector $\mathbf{a}\in\{-r,-r+1,\dots,r\}^m$ the ciphertext is: $(\mathbf{u}=\mathbf{A}^T\mathbf{a},\mathbf{c}=\mathbf{P}^T\mathbf{a}+f(\mathbf{v}))\in\mathbb{Z}_q^n\times\mathbb{Z}_q^l$.

- The decryption of the ciphertext $(\mathbf{u},\mathbf{c})$ is computed using the private key $\mathbf{S}\in\mathbb{Z}_q^{n\times l}$, so you can recover the plaintext as $f^{-1}(\mathbf{c}-\mathbf{S}^T\mathbf{u})$.

The decryption is given by the following simple mathematical operations:

$$\begin{aligned} f^{-1}(\mathbf{c}-\mathbf{S}^T\mathbf{u}) &= f^{-1}(\mathbf{P}^T\mathbf{a}+f(\mathbf{v})-\mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ &= f^{-1}((\mathbf{A}\mathbf{S}+\mathbf{E})^T\mathbf{a}+f(\mathbf{v})-\mathbf{S}^T\mathbf{A}^T\mathbf{a}) = f^{-1}(\mathbf{E}^T\mathbf{a}+f(\mathbf{v})) \end{aligned} \tag{3.13}$$

As can be seen from the last passage, if the value of a coordinates $\mathbf{E}^T\mathbf{a}$ is greater then the half of the value that the function f is supposed to round this will introduce an error in the decoding. We can address with this issue with the right choice of the parameters or maybe using an error correcting code to encode the message before encrypt it.

The security of this crypto-system is mainly addressed by the supposed hard problem to distinguish between public keys $(\mathbf{A},\mathbf{P})$ and uniformly chosen at random pair of matrix. From this assumption follow also that if one tries to encrypt a message with a random public key he is not able to recover any statistical information about the encrypted message.

3.3 Multivariate Public Key crypto-systems

Multivariate (Public-Key) Cryptography is the study of PKCs where the trapdoor one-way function takes the form of a multivariate quadratic polynomial map over a finite field [64, 52]. Namely the public key is in general given by a set of quadratic polynomials:

$$\mathcal{P}=(p_1(w_1,\dots,w_n),\dots,p_m(w_1,\dots,w_n)), \tag{3.14}$$

where each p_i in the most modern construction is of the type:

$$z_k=p_k(\mathbf{w}):=\sum_i P_{ik}w_i+\sum_i Q_{ik}w_i^2+\sum_{i>j}R_{ijk}w_iw_j \tag{3.15}$$

with all coefficients and variables in $\mathbb{F}_q$, the field with q elements. The evaluation of these polynomials at any given value corresponds to either the encryption procedure or the signature verification procedure. Inverting a multivariate quadratic map is equivalent to solving a set of quadratic equations over a finite field, i.e. solve the system $p_1(\mathbf{x})=p_2(\mathbf{x})=\cdots=p_m(\mathbf{x})=0$. All coefficients and variables are in $\mathbb{F}_q$, the field with q elements.This is referred as the $\mathcal{MQ}$ problem, and a general instance of this problem (with a random set of polynomial) is NP-hard. Of course a general version of this problem cannot hide a trapdoor, so couldn't be used in a MPKC, anyway solve a specific set of polynomial (in which is possible to hide a trapdoor) is still believed to be an hard problem.

3.3.1 Basic Constructions

A MPKC is slightly different from the other two families of crypto-system described so far, because a MPKC can be build only in fewer way than the others but there are a lot of variance where the only thing that change is the way to hide the private map (i.e. making computational hard find it from the public map just enlarging the number of polynomials in the public key, adding some random polynomials or removing some polynomials to make the equations, relative to the public map, underdetermined). So it will be more useful to provide a brief description of the basic constructions of a MPKC [10].

The Standard Construction

The way to hide the trapdoor is not unique, however, the MPKCs almost always hide the private map $\mathcal{Q}$ via composition with two affine maps S, T. So, $\mathcal{P} = T \circ Q \circ S : \mathbb{K}^n \to \mathbb{K}^m$, or

$$\mathcal{P} : \mathbf{w} = (w_1, \ldots, w_n) \overset{S}{\to} \mathbf{x} = M_S\mathbf{w} + \mathbf{c}_S \overset{\mathcal{Q}}{\to} \mathbf{y} \overset{T}{\to} \mathbf{z} = M_T\mathbf{y} + \mathbf{c}_T = (z_1, \ldots, z_m) \quad (3.16)$$

In any given scheme, the central map $\mathcal{Q}$ belongs to a certain class of quadratic maps whose inverse can be computed relatively easily. The maps S, T are affine (sometimes linear) and full-rank.

- The *public key* consists of the polynomials in $\mathcal{P}$. In practice, this is always the collection of the coefficients of the $p_i s$, compiled in some order conducive to easy computation;
- the *secret key* consists of the informations in S, T and $\mathcal{Q}$. That is, we collect $(M_S^{-1}, \mathbf{c}_S), (M_T^{-1}, \mathbf{c}_T)$ and whatever parameters there exist in $\mathcal{Q}$;
- to verify a *signature* or to *encrypt* a block, one simply computes $\mathbf{z} = \mathcal{P}(\mathbf{w})$;
- to *sign* or to *decrypt* a block, one computes $\mathbf{y} = T^{-1}(\mathbf{z}), \mathbf{x} = Q^{-1}(\mathbf{y})$ and $\mathbf{w} = S^{-1}(\mathbf{x})$ in turn.

Implicit Form MPKCs

In this construction the public key is a system of l equations

$$\mathcal{P}(\mathbf{w},\mathbf{z}) = \mathcal{P}(w_1,\ldots,w_n,z_1,\ldots,z_m) = (p_1(\mathbf{w},\mathbf{z}),\ldots,p_l(\mathbf{w},\mathbf{z})) = (0,\ldots,0),$$

where each p_i is a polynomial in $\mathbf{w} = (w_1,\ldots,w_n)$ and $\mathbf{z} = (z_1,\ldots,z_m)$. This $\mathcal{P}$ is built from the secret $\mathcal{Q}$

$$\mathcal{Q}(\mathbf{x},\mathbf{y}) = q(x_1,\ldots,x_n,y_1,\ldots,y_m) = (q_1(\mathbf{x},\mathbf{y}),\ldots,q_l(\mathbf{x},\mathbf{y})) = (0,\ldots,0),$$

where $q_i(\mathbf{x},\mathbf{y})$ is polynomial in $x = (x_1,\ldots,x_n)$, $y = (y_1,\ldots,y_m)$ such that:

- For any given specific element x', we can easily solve the equation:

$$Q(\mathbf{x}',\mathbf{y}) = (0,\ldots,0);$$

- for any given specific element y', we can easily solve the equation:

$$Q(\mathbf{x},\mathbf{y}') = (0,\ldots,0),$$

Now, we can build

$$\mathcal{P} = L \circ h(S(\mathbf{w}),T^{-1}(z)) = (0,\ldots,0),$$

where S,T are invertible affine maps and L is linear. To verify a signature $\mathbf{w}$ with the digest $\mathbf{z}$, one checks that $P(\mathbf{w},\mathbf{z}) = 0$. If we want to use $\mathcal{P}$ to encrypt the plaintext $\mathbf{w}$, we would solve $P(\mathbf{w},\mathbf{z}) = (0,\ldots,0)$, and find the ciphertext $\mathbf{z}$.

To invert (i.e., to decrypt or more likely to sign) $\mathbf{z}$, one first calculates $\mathbf{y}' = T^{-1}(\mathbf{z})$, then plugs $\mathbf{y}'$ into the equation $Q(\mathbf{x}',\mathbf{y}') = (0,\ldots,0)$ and solve for $\mathbf{x}$. The result plaintext or signature is given by $\mathbf{w} = S^{-1}(\mathbf{x})$. To recap, in an implicit-form MPKC, the public key consists of the l polynomial components of $\mathcal{P}$ and the field structure of $\mathbb{K}$. The secret key mainly consists of L,S and T. Depending on the case the equation $Q(\mathbf{x},\mathbf{y}) = (0,\ldots,0)$ is either known or has parameters that is a part of the secret key.

Again the basic idea is that S,T,L serve the purpose to "hide" the equation $Q(\mathbf{x},\mathbf{y}) = 0$, which otherwise could be easily solved for any y.

Isomorphism of Polynomials

The IP problem originated by trying to attack MPKCs by finding the secret keys. Let F_1 and F_2 be two polynomials maps from $\mathbb{K}^n \to \mathbb{K}^m$. The IP problem is to look for two invertible affine linear transformations $S \in \mathbb{K}^n$ and T over $\mathbb{K}^m$ (if they exist) such that:

$$F_1(x_1, \dots, x_n) = T \circ F_2 \circ S(x_1, \dots, x_n).$$

It is clear that this problem is closely related to the attack of finding private keys for a MPKC.

Basically the logic behind the MPKC is all expressed by these three construction. It is easy to understand that each MPKC is fully described by the central map and there is not so much freedom in the building of a crypto-system. Of course there are some empirical refinement that can be applied to better hide the trapdoor and enhance security (at the cost of slowing down encryption or signature or both) or to expedite computation that can more or less be summarized in:

- add extra random polynomials in the central map ($\mathcal{Q}$ with our notation) that enhances security but slows the signatures
- remove central or public polynomials that enhances security but slows the encryption
- add internal perturbations (that means to add to the central map a random number of function $f(\mathbf{v})$, where $\mathbf{v}$ is an r-tuple of random affine forms) that again is a better concealment for the central map but slows everything
- add extra variables that can be set arbitrarily that slow encryption
- make the central map sparse (that means to set to 0 a large number of coefficients), this can be a reduction in the security but induce a general speedup.

3.4 Framework

In this section I would like to provide a set of parameters on which we can base the choice of an algorithm for further studies and implementation in first place.

A first categorization of parameters can be made on the base of a theoretical view point of view and/or on a practical one.

The *theoretical parameters* that can be taken in consideration so far are:

- the algorithmic hardness of the solution for problem upon which is based the trapdoor;

- hints about the weakness against the two known quantum algorithm;
- vulnerabilities to the different model of attacks (i.e. CPA. CCA. CCA2, etc.);

The *practical parameters* taken in consideration so far are:

- RSA-style signature;
- public and private key length, blow-up factor for the ciphertext;
- number of operation per encryption and decryption, signing and signature evaluation;
- complexity of the operations per encryption and decryption, signing and signature evaluation;
- memory needed for the operations;
- oldness of the algorithmic problem and trust on it;

Except from this two small list of parameters the first discriminant used for the choice of the algorithm is the capability to perform encryption, for this reason I didn't examinate the "Hash-based Digital Signature Schemes" [10] in the first time, although the capability to easily perform a signature within the same family of crypto-system has to be taken into account.

3.4.1 Algorithmic Problems

- The Code Based crypto-systems viewed in this paper are security equivalent. The security of a PKC based on error correcting code depends on two assumption: the hardness of decoding in a random linear code and the indistinguaishability of the code upon which the crypto-system is based. The first is an old problem of coding theory for which only exponential time solution are known. The second is not so old but is conjectured to be hard for the Goppa code.
- The Lattice based crypto-systems have no theoretical proof of security except for the LWE-based one. Their security is based on the presumability hardness of the already defined lattice problems. So far olny exponential time algorithm are known to recover the shortest vector in a lattice[9] or to diagonalize[10] a lattice basis. The

[9]The solution of this problem will drive to the recovery of the message and can be made harder with a proper parameter choice has to be done

[10]This mean the ability to recover the private key from the public key. So far only the LLL algorithm [34] is the one with the best performance, but after the fourth dimension is not feasible any more.

LWE is related to the learning parity with noise problem which is believed to be very hard and is also closely related to decoding problems in coding theory. The LWE was shown to have a worst-case connection to the approximate-SVP and SIVP. This reduction however is a quantum reduction [55]. In other words, breaking the crypto-system implies an efficient quantum algorithm for approximating the SVP problems.

- The MPKC hardness is derived from the hardness of the general $\mathcal{MQ}$ problem, that is: solve a random system of polynomials $p_1(\mathbf{x}) = p_2(\mathbf{x}) = \cdots = p_m(\mathbf{x}) = 0$ where each polynomial is quadratic in $\mathbf{x}$ and all coefficient are in $\mathbb{K} = \mathbb{F}_q$ is NP-hard. Of course the set of polynomial cannot be random if we want to use them to hide a trapdoor, so we don't have the theoretical proven hardness but the problem is still believed to be hard with the proper choice of parameters.

Except from the LWE construction we can trust these PKC only based on the "oldness" of the problem upon which is based the trapdoor: i.e. there is no theoretical proof of security for none of the know PKC, we can only trust them in the sense that they are old standing problems and nobody found a good attack until now.

3.4.2 Quantum Algorithm Weakness

The only known and exploitable algorithms to break the current state of the art PKI are the Shor's and Grove's algorithms. The first is responsible to the accomplish the factorization in polynomial time, that leads to the solution of the RSA problem in polynomial time too, and the second can speed up the search for an element in an unsorted list by a quadratic factor. The PKCs introduced so far are supposed to be insensitive to the Shor's algorithm, although the Lattice based crypto-system shows a "certain amount of periodicity" (intrinsic in the lattice definition) that could be exploited, but there are no know quantum algorithm capable of such exploit yet. On the other hand the Grover's algorithm can be used to make faster a brute force attack, i.e. it can be a treat for all crypto-system, even the symmetric ones. This algorithm anyway cannot speed up the search in an incisive way, so it can be taken into account just enlarging the searching space making a proper parameter choice.

3.4.3 Attack model vulnerabilities

The current constructions of the code based PKC are CCA secure, the lattice based are only CPA secure, even if there are few versions of the lattice based crypto-systems in the literature that appears to enjoy the CCA security too.

3.4.4 RSA-style signature

Although the capability to perform digital signatures is not the most important discriminant it allow us to make a choice: indeed the only crypto-systems that allow to easily perform a signature are the lattice-based ones and the MPKCs. Within the code based crypto-systems were build only an interactive identification protocol (Stern ID protocol) [20] from which is possible to derive a signature procedure but it is quite expensive(it is based on a challenge with a cheating probability of arround 2/3 so it needs to be repeated a large number of times). The lattice based PKC allow to build a signature scheme quite easily, there is an example: the “NTRUsign” but it was demonstrated that this signature leak a lot of information about the private basis, so to take in account the leakage we need to apply a perturbation to it, slowing down the procedure [49]. Indeed only the MPKCs allow a very simple signature protocol that is really similar to RSA style of signature.

3.4.5 Dimension and complexity

All of the crypto-system presented so far have the same drawback on the side of the memory consumption: all of them have really large public keys. The dimensions can be reduced exploiting special rings in which define the algebra [39, 40, 21, 17, 37], but the ”structured” PKC couldn’t enjoy the same security proofs or reductions to other well known problem indeed they are ”supposed” to be secure like the full-size counterparts. We always need to be aware of the security weaknesses that could come from it. There are a lot of ways to reduce the memory consumption depending on each particular construction. At this stage it has not so much sense to list all the known ones.

The major advantage on the other side is that the crypto-system involve only simple operations (i.e. addiction, multiplication and reduction modulo compared to the exponentiation and reduction modulo of the RSA) to perform encryption/signature verification and decryption/signing. These operations are in general much more simple than the RSA-style crypto-systems. So the implementation of such a crypto-systems allow to perform the same tasks of the RSA ones in a faster way.

3.5 Reasons to choose the lattice based PKC

The most interesting crypto-systems family, according to the chosen parameters, is the *Lattice-Based* one. There are different reasons to conclude that:

1. we can build crypto-systems that share security assumption both from the code based (i.e. see the GGH crypto-system) and in a more large sense form MPKC too(a system of polynomials is written as the matrix of the coefficients, so find the shortest vector in a lattice could be seen as to find a small solution for the associated system of polynomials);

2. this crypto-systems family allow to build a simple signature protocol, this is not the case of the Code Based family;

3. the MPKC can exploit a poor variety of mathematical problems, they all rely on the $\mathcal{MQ}$ problem. The various constructions differ in the number of polynomials or variables used and in the dimensions or cardinality of the field in which the polynomials are define;

4. the attempt to reduce the memory consumption, required for the public key for instance, do not influence so much the security or the computation speed of the various tasks as the case of the MPKCs;

5. [11] a recent breakthrough in the cryptology led to the formulation of a new way of thinking about encryption: the homomorphic encryption [48]. This is a form of encryption that allows to carry out computations on ciphertext directly, without needing for decryption and subsequent re-encryption of the results. The LWE PKC is a natural candidate for it!

These points define the framework upon which I based my choice of investigate further the lattice-based family of PKCs. I could also include as personal reason my prior knowledge of the topic that is closely related to my studies, and to the physics world, and therefore already familiar at the beginning of the study.

[11] This is nothing more than a curiosity at this stage

Chapter 4

A closer look at the lattice based PKC

It is interesting, at this stage, to have an overview of the encryption processes of the three most popular lattice PKC construction. I will provide a picture only of the basic constructions of the lattice-baed PKC. The idea is to provide a high level picture through simple numeric examples of the different encryption and decryption processes.

4.1 Introduction

Informally speaking a lattice is a periodic collection of points described by a basis, which means it can be viewed like the span of the column vectors of basis in which the coefficients are all integers.

One of the most important mathematical features of the lattice constructions is that a lattice has an infinite number of bases: this is one of the most important feature on which the security of the crypto-systems can rely on.

4.2 The GGH crypto-system

The GGH crypto-system is the only one entirely based on a lattice construction and can be seen as a lattice variant of the McElice proposal of the Code Based crypto-systems.

The basic idea is to encode the message in a lattice point and add to it some random noise in order to encrypt it. An equivalent construction is to encode the message in the "noise" that would not be random any more and add it to a random lattice point.

The decryption process take place removing the noise[1] from the lattice point through

[1]The noise needs to be bound in order to be correctly removed and the value is related to the dimensions

a rounding procedure that uses the "private key". In the following we will consider only the approach in which the message is encoded in the noise because the two constructions are equivalent on the mathematical and security point of view.

4.2.1 Keys generation

The private key in the GGH crypto-system is a "good basis" R with n vectors $v \in \mathbb{Z}^n$ that is an "almost rectangular lattice" basis. It is built in this way:

$$R \leftarrow R' + kI \tag{4.1}$$

where R' is uniformly distributed in $\{-l, \dots, l\}^n$ and l is some small integer, k is an integer and I is the identity matrix.

The public basis B can be generated starting from the private one and applying a series of Gaussian operations in order to mix and hide the "almost rectangular" structure.

For a numeric example we can choose $l = 4$ and $k = 20$ and $n = 5$ in order to obtain:

$$R = \begin{bmatrix} -1 & -2 & 1 & -2 \\ -3 & -4 & 1 & -4 \\ 4 & 0 & 3 & -1 \\ -3 & 2 & 3 & -3 \end{bmatrix} + \begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix}$$

Computing the sum we will obtain our private matrix R. It is easy to notice the "almost rectangular" structure of the matrix.

$$R = \begin{bmatrix} 19 & -2 & 1 & -2 \\ -3 & 16 & 1 & -4 \\ 4 & 0 & 23 & -1 \\ -3 & 2 & 3 & 17 \end{bmatrix} \tag{4.2}$$

From the private matrix R we can obtain the public one by mixing its rows. An idea is to add to each row a linear combination of the other rows with coefficients chosen between $\{-1, 0, 1\}$.

A more recent proposal is to use the **Hermite Normal Form** of the matrix because it doesn't depend on the particular choice of the matrix but it is a lattice invariant. So it

of the coefficients of the private basis.

does not leak any information about the private basis.[2]

The HNF can be computed using Gaussian operation and it is defined as a triangular matrix where the coefficients on the diagonal are all bigger than the ones in the corresponding column.

Computing the HNF of R we obtain our public matrix B that is another basis of the same lattice:

$$B = \begin{bmatrix} 1 & 0 & 0 & 30340 \\ 0 & 2 & 0 & 2839 \\ 0 & 0 & 1 & 9947 \\ 0 & 0 & 0 & 58357 \end{bmatrix} \tag{4.3}$$

4.2.2 Encryption and Decryption

To perform the encryption we choose a random lattice point and we add to it the “noise” in which the message is encoded. The random lattice point can be generated by the multiplication of the lattice basis B with a random vector of coefficients $x \in \mathbb{Z}^m$. Unfortunately this operation is not computationally feasible, because it involves the generation of hundreds of ”big” random number that is in general a very expensive operation.

An useful approach to solve this issue is to notice that every lattice induces an equivalence relation over $\mathbb{Z}^n$ defined as: $v \equiv_L w$ iff $v - w \in L$. We can think to build a lattice subspace where the reduced lattice point w is unique: the *orthogonalized parallelepiped* $P(B^*) = \{\sum_i x_i \mathbf{b_i^*} | 0 \leq x_i \leq 1\}$. To compute w we can project the component of the vector v on the orthogonalized basis [3] and then subtract from v the projected components of the same vector multiplied the basis vectors.

$$\mathbf{w} = \mathbf{v} - \sum_i \left\lfloor \frac{\langle \mathbf{v}, \mathbf{b_i^*} \rangle}{\langle \mathbf{b_i^*}, \mathbf{b_i^*} \rangle} \right\rfloor \mathbf{b_i} \tag{4.4}$$

Since w and v differs only by integer multiplies of the basis vectors the equivalence $v \equiv_L w$ holds.

The unique element $w \in P(B^*)$ is denoted $v \pmod B$. This construction is “almost equivalent” to adding a random lattice point to the noise and reducing it.

[2]This is not the case when we choose as public basis the mixed version of the private one. The private can be recovered using a reduction algorithm like the LLL [34], so using the HNF make its usage far more difficult.

[3]The orthogonalized basis obtained from one basis in the HNF form is simply the diagonal matrix with entries the $b_{i,i}$ from the previous one.

In other words we can compute $c = Bx + r$ and reduce it $mod\, B$ but Bx is a lattice point so it belongs to the equivalence class of $[0]_L$ in the orthogonalized parallelepiped $P(B^*)$. So after the reduction only the $r\, mod\, B$ part of the cryptogram would survive. Indeed the reduction modulo makes the cryptanalysis harder because the reduction can be computed starting from any random lattice point.

For example we can choose and error vector $r = \{10, 10, 10, 10\}$ and encrypt it reducing it modulo the public basis B using the equation 4.4.

$$c = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 10 \end{bmatrix} \pmod{\begin{bmatrix} 1 & 0 & 0 & 30340 \\ 0 & 2 & 0 & 2839 \\ 0 & 0 & 1 & 9947 \\ 0 & 0 & 0 & 58357 \end{bmatrix}} = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 10 \end{bmatrix} \tag{4.5}$$

The decryption corresponds to finding the lattice point v closest to the target ciphertext c and compute $r = c - Bv$. To perform this task we can use the Babai's Round-off algorithm [3]. The idea behind this algorithm is purely geometrical.

We can see the cryptogram c as a linear combination of the columns of the private basis R, i.e.

$$c = R(\sum_i \alpha_i \hat{e}_i) \tag{4.6}$$

Rounding these coefficients to the closest integers we obtain a lattice point.

$$\lceil R^{-1} c \rfloor = \left\lceil \sum_i \alpha_i \hat{e}_i \right\rfloor \tag{4.7}$$

We can safely make the assumption that this lattice point is the Bv that we are looking for:

$$Bv = R\lceil R^{-1} c \rfloor \Rightarrow v = B^{-1} R \lceil R^{-1} c \rfloor \tag{4.8}$$

From v we can obtain $r = c - Bv$.

With this choice of parameters the decryption is expected to fail with high probability. Within this small example the the decryption is: $\mathbf{v} = [2, 9, 1, -7]^T$ that is completely wrong.

Running few other encryption/decryption cycles I always got a zero vector as ciphertext if I chose smaller "noises". In this example the noise is the message. The problem is that with such small matrices I need to choose "big" entries in order to obtain a small norm of the rows of the inverse of the private basis. Apparently it is still not enough to encode a usable noise, i.e. in which we could encode a message. The norm of the inverse matrix's rows is a boundary to the probability of making errors in the decoding, this is why its rows have to be small.

This is of course an unnatural way of using the crypto-system but it helps us to highlight the algebraic difficulties behind it.

Another implementation problem is that the matrices inverse need to have entries with a large number of digits after the "decimal point" and any too crude approximation, i.e. cutting too many digits after the decimal point, induces decryption errors.

We can see that even if this is one of the most natural lattice based crypto-system, it is indeed quite complicate and in general not well defined, on a mathematical point of view, how to realize the different functions that make the crypto-system.

4.3 The NTRU crypto-system

This crypto-system is build in the frame of the truncated polynomial ring $R[X]/(X^N - 1)$ where all the polynomials have integer coefficients with degree at most $N - 1$. The NTRU can be built in the truncated polynomial ring since it is completely equivalent to the convolutional lattice frame, where it can be also equivalently described as an instance of the general GGH considering the encryption process again as a reduction modulo a basis.

In order to understand the relationship between the two spaces we can see how the multiplication between a vector and a matrix resembles to the multiplication between two polynomials in the ring $R = Z_q[X]/(X^N-1)$. The product of the polynomials $a(X) \times b(X)$, where $a(X)$ and $b(X) \in R$, is equivalent to the discrete cyclic convolution of the vectors containing the coefficients of the two polynomials.

We can view $\mathbf{a}(X)$ as the vector $\mathbf{a} = [a_0, a_1, \ldots, a_{N-1}]$ as well as $\mathbf{b}(X)$ as the vector $\mathbf{b} = [b_0, b_1, \ldots, b_{N-1}]$, where the higher degree terms are encoded in the first position of the vector, i.e. $x^{N-1} \to a_0$. I adopted this unusual encoding to simplify the explanation of the algorithm but it could be changed properly relabeling the matrix elements.

The cyclic convolution, between two vectors $\mathbf{a}$ and $\mathbf{b}$, is defined like the scalar product between the matrix $\mathbf{T}^*\mathbf{a}$ and the vector $\mathbf{b}$, where the matrix $\mathbf{T}^*\mathbf{a}$ is obtained from $\mathbf{a}$ cyclically shifting its coordinates:

$$c(X) = a(X) * b(X) = \begin{bmatrix} a_{N-1} & a_{N-2} & \dots & a_0 \\ a_0 & a_{N-1} & \dots & a_1 \\ a_1 & a_0 & & \vdots \\ \vdots & \vdots & \ddots & \\ a_{N-2} & a_{N-3} & \dots & a_{N-1} \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ b_{N-1} \end{bmatrix}$$

.

The result of this matrix vector multiplication is a vector equivalent to a polynomial in the truncated ring. For instance $(x+1) \cdot x$ in the ring $\mathbb{Z}_2/(x^3-1)$ is equivalent to the convolution between $[0,1,1]$ and $[0,1,0]$ defined as:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

.

The result, re-decoded in the truncated polynomial ring using the same conversion rules, is equivalent to $x^2 + x$, as expected.

4.3.1 Keys generation

The private key is a short vector made of two vector: $\mathbf{f}$ and $\mathbf{g}$ that define the key: $(\mathbf{f}, \mathbf{g}) \in \mathbb{Z}^{2n}$. These two vectors are subject to the restriction:

- $\mathbf{f} \in \hat{e}_1 + \{-p, 0, p\}^n$,
- $\mathbf{g} \in \{-p, 0, p\}^n$,
- the matrix $[T^*\mathbf{f}]$, whose columns are a copy of the vector $\mathbf{f}$ but with rotated coordinates [4], must be invertible modulo q.

We can build a simple example to use like a toy model in order to understand the basic algebraic operations. We set $p = 3$ and $q = 255$, one possible choice for the private vectors

[4]The column vectors of the matrix $[T^*\mathbf{f}]$ are defined as: $\mathbf{f_1} = \{f_1, f_2, \dots, f_n\}$, $\mathbf{f_2} = \{f_n, f_1, \dots, f_{n-1}\}$, $\dots$, $\mathbf{f_n} = \{f_2, f_3, \dots, f_1\}$.

is: $\mathbf{f} = [1, 3, 3, -3, 0]^T$ and $\mathbf{g} = [3, 3, -3, 0, 0]^T$. Applying the T^* operator to $\mathbf{f}$[5] we obtain:

$$[T^*\mathbf{f}] = \begin{bmatrix} 1 & 0 & -3 & 3 & 3 \\ 3 & 1 & 0 & -3 & 3 \\ 3 & 3 & 1 & 0 & -3 \\ -3 & 3 & 3 & 1 & 0 \\ 0 & -3 & 3 & 3 & 1 \end{bmatrix}$$

The restriction on this matrix is that is has to be invertible modulo q. The inverse will be:

$$[T^*f]^{-1} = \begin{bmatrix} 181 & 24 & 27 & 111 & 231 \\ 231 & 181 & 24 & 27 & 111 \\ 111 & 231 & 181 & 24 & 27 \\ 27 & 111 & 231 & 181 & 24 \\ 24 & 27 & 111 & 231 & 181 \end{bmatrix} \mod 255$$

The public key is once again the HNF of the basis of the lattice built starting from the private vector. The lattice used by this crypto-system is the <u>convolutional modular lattice</u>. That is defined as the m-dimensional lattice:

$$\Lambda_q(A) = \Big\{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y} = \mathbf{A}^T\mathbf{s} \bmod q \text{ for some } \mathbf{s} \in \mathbb{Z}^n\Big\}$$

for some $A \in \mathbb{Z}_q^{n\times m}$ with $n \leq m$.

It's straightforward to see that the lattice is generated by the vectors $\mathbf{y} \in \mathbb{Z}^m$, but according to the definition we get only n vectors that are less then the dimension of the lattice. To completely define the basis we exploit "the modulo equivalence" defining the missing $m - n$ vectors as: $y_i = q\hat{e}_i$.

We can feed Λ_q with our cyclic matrix $(T^*\mathbf{f}, T^*\mathbf{g})^T \in \mathbb{Z}^{n\times 2n}$ and obtain the basis:

$$B = \begin{bmatrix} [T^*\mathbf{f}]^T_{n\times n} & \mathbf{0}_{n\times n} \\ [T^*\mathbf{g}]^T_{n\times n} & q\mathbf{I}_{n\times n} \end{bmatrix}$$

HNF form of this basis has the following structure:

$$H = \begin{bmatrix} \mathbf{I}_{n\times n} & \mathbf{0}_{n\times n} \\ [T^*\mathbf{h}]_{n\times n} & q\mathbf{I}_{n\times n} \end{bmatrix}$$

[5]For the vector $\mathbf{g}$ is the same procedure.

where $\mathbf{h} = [T^*\mathbf{f}]^{-1}\mathbf{g} \bmod q$. All the information related to the public basis is in the vector $\mathbf{h}$ so we can safety keep only this vector as public basis. Following our example we get $\mathbf{h} = [24, 144, 228, 231, 75]^T$

4.3.2 Encryption and Decryption

The encryption as for the GGH is the reduction modulo the public basis H of a vector somehow encoding the message. To perform the reduction see equation 4.4. An example is to pick up a message plus a random padding vector $(-\mathbf{r}, \mathbf{m}) \in \{1, 0, -1\}^{2n}$, this yields to:

$$\begin{bmatrix} -\mathbf{r} \\ \\ \mathbf{m} \end{bmatrix} \bmod \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \\ [T^*\mathbf{h}] & q\mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \\ (\mathbf{m} + [T^*\mathbf{h}]\mathbf{r}) \bmod q \end{bmatrix} \tag{4.9}$$

The cyphertext is $\mathbf{c} = (\mathbf{m} + [T^*\mathbf{h}]\mathbf{r}) \bmod q$.

For a simple example we can choose $\mathbf{m} = [1, 1, 0, 0, -1]^T$ to be the message and $\mathbf{r} = [1, 0, 0, -1-1]^T$ to be the random padding. The cryptogram is computed from the equation 4.9:

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} + \begin{bmatrix} 24 & 75 & 231 & 228 & 144 \\ 144 & 24 & 75 & 231 & 228 \\ 228 & 144 & 24 & 75 & 231 \\ 231 & 228 & 144 & 24 & 75 \\ 75 & 231 & 228 & 144 & 24 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ -1 \end{bmatrix} \bmod 255$$

$$= \begin{bmatrix} 163 \\ 196 \\ 177 \\ 132 \\ 161 \end{bmatrix}$$

To decrypt $\mathbf{c}$ it's enough to multiply it on the left by $[T^*\mathbf{f}] \bmod q$ and reduce the product modulo p. The decryption rely on:

$$[T^*\mathbf{f}]\mathbf{c} \bmod q = [T^*\mathbf{f}]\mathbf{m} + [T^*\mathbf{f}][T^*\mathbf{h}]\mathbf{r} \bmod q = [T^*\mathbf{f}]\mathbf{m} + [T^*\mathbf{g}]\mathbf{r} \bmod q$$

Once again using the numbers the last piece of the equation is:

$$[T^*\mathbf{f}]\mathbf{c} \bmod 255 = \begin{bmatrix} 1 & 0 & -3 & 3 & 3 \\ 3 & 1 & 0 & -3 & 3 \\ 3 & 3 & 1 & 0 & -3 \\ -3 & 3 & 3 & 1 & 0 \\ 0 & -3 & 3 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 163 \\ 196 \\ 177 \\ 132 \\ 161 \end{bmatrix} \bmod 255$$

$$= \begin{bmatrix} 1 \\ 7 \\ 6 \\ 252 \\ 245 \end{bmatrix}$$

Where the coordinate of $[T^*\mathbf{f}]\mathbf{m} + [T^*\mathbf{g}]\mathbf{r} \bmod q$ are bounded by $q/2$, so the reduction modulo q doesn't change any coordinate of the vector. Reducing the last term modulo p we recover the message $\mathbf{m}$ thank to the restrictions on $\mathbf{f}$ and $\mathbf{g}$, i.e.:

$$\mathbf{m} = \begin{bmatrix} 1 \\ 7 \\ 6 \\ 252 \\ 245 \end{bmatrix} \bmod 3 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 2 \to -1 \end{bmatrix}$$ [6]

4.4 The Learning with Errors crypto-system

The LWE is an efficient crypto-system and very suitable for optimization. As the the NTRU it can be built in a lattice framework but it shares a lot of features with the code correction problems too. The idea that leads to the security of the crypto-system is that solving a system of linear equations with errors, with a distribution χ, is considered difficult.

[6] $-1 \bmod 3 \equiv 2 \bmod 3$ but $\mathbf{m} \in \{-1, 0, 1\}$ so we can safety assume that each 2 is a -1 without committing any decryption error.

Given a set of equations of the type:

$$\begin{aligned}
\langle \mathbf{s},\mathbf{a}_1\rangle &\approx_\chi b_1 (\text{mod } q)\\
\langle \mathbf{s},\mathbf{a}_2\rangle &\approx_\chi b_2 (\text{mod } q)\\
&\vdots\\
\langle \mathbf{s},\mathbf{a}_n\rangle &\approx_\chi b_n (\text{mod } q)
\end{aligned}$$

Where $\langle \cdot, \cdot \rangle$ is the scalar multiplication between two vectors, $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{a_i}$ is chosen independently from the uniform distribution on $\mathbb{Z}_q^n$ and $b_i \in \mathbb{Z}_q$. And each equation is independently correct with probability $1 - \chi$, therefore each error is sampled according to an error distribution χ. The problem take as input the couples $(\mathbf{a}_i, b_i)$ and the goal is to determinate $\mathbf{s}$. The b_is have the form: $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$ where each $e_i \in \mathbb{Z}_q$ is independently chosen according to the distribution χ. Looking at the equations it is clear that the LWE is equivalent to the problem of decoding random linear codes.

The crypto-system is quite simple, we first need a simple error-tolerant encoder and decoder that map the message space to another one suitable for encryption purposes, i.e. $f : \Sigma \to \mathbb{Z}_q^l$.

One example is to choose an encoder that add a random noise bounded by half of the value of the modulo, i.e. $\lfloor \frac{q}{2} \rfloor$ to encrypt a bit 1 and doesn't add anything to encrypt a 0.

<u>We will use the encoder</u> $f : \mathbb{Z}_t^l \to \mathbb{Z}_q^l$ that map the message space simply multiplying each coordinate by q/t and rounding it to the nearest integer. The inverse can be defined simply with a function that divide each coordinate by q/t rounding it to the nearest integer too.

4.4.1 Keys generation

The keys are quite simple to generate. The private key is a matrix $\mathbf{S} \in \mathbb{Z}_q^{n \times l}$ with all the entries uniformly chosen at random.

To provide a numerical example we can fix the parameters to small values: $n = 5, l = 4, m = 3, t = 5, q = 255, \alpha = 0.01$. The private key is chosen at random with the entries between 0 and 254.

$$\mathbf{S} = \begin{bmatrix} 211 & 13 & 112 & 111 \\ 192 & 36 & 105 & 173 \\ 83 & 100 & 232 & 20 \\ 37 & 60 & 39 & 225 \\ 65 & 53 & 68 & 159 \end{bmatrix}$$

To generate the public key we need to choose the matrices: $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, uniformly at random, and $\mathbf{E} \in \mathbb{Z}_q^{m \times l}$ with each entry chosen according to a probability distribution, an example is a normal distribution with zero mean and a STD proportional to q[7]. The public key is just: $(\mathbf{A}, \mathbf{P} = \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times l}$. A simple numeric example is:

$$\mathbf{A} = \begin{bmatrix} 61 & 96 & 224 & 151 & 100 \\ 57 & 78 & 231 & 102 & 86 \\ 141 & 29 & 187 & 188 & 91 \end{bmatrix}$$

$$\mathbf{E} = \begin{bmatrix} 0 & 254 & 0 & 0 \\ 254 & 1 & 254 & 0 \\ 254 & 0 & 2 & 0 \end{bmatrix}$$

$$\mathbf{P} = (\mathbf{AS} + \mathbf{E}) \bmod 255 = \begin{bmatrix} 17 & 208 & 224 & 214 \\ 204 & 98 & 216 & 120 \\ 215 & 195 & 8 & 87 \end{bmatrix}$$

4.4.2 Encryption and Decryption

To perform the encryption we choose a message $\mathbf{v}$ from the message space Σ, a public key and a random vector $\mathbf{a} \in \{-r, -r+1, \ldots, r\}^m$. The ciphertext is the couple of vectors:

$$(\mathbf{u} = \mathbf{A}^T \mathbf{a}, \mathbf{c} = \mathbf{P}^T \mathbf{a} + f(\mathbf{v})) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^l$$

We choose the random $\mathbf{a} = [5, 8, -7]^T$ and the message $\mathbf{v} = [2, 0, 3, 1]^T \in \mathbb{Z}_5^4$. From

[7]From the standard deviation of the error distribution follows the probability of decryption errors. In our numeric example we take it very small, it means that the errors are very close to zero.

this choice it follows: $f(\mathbf{v}) = [102, 0, 153, 51]^T$ and the ciphertext:

$$\mathbf{u} = \begin{bmatrix} 61 & 96 & 224 & 151 & 100 \\ 57 & 78 & 231 & 102 & 86 \\ 141 & 29 & 187 & 188 & 91 \end{bmatrix}^T \times \begin{bmatrix} 5 \\ 8 \\ -7 \end{bmatrix} \mod 255 = \begin{bmatrix} 29 \\ 136 \\ 129 \\ 0 \\ 41 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 17 & 208 & 224 & 214 \\ 204 & 98 & 216 & 120 \\ 215 & 195 & 8 & 87 \end{bmatrix}^T \times \begin{bmatrix} 5 \\ 8 \\ -7 \end{bmatrix} + \begin{bmatrix} 102 \\ 0 \\ 153 \\ 51 \end{bmatrix} \mod 255 = \begin{bmatrix} 59 \\ 204 \\ 140 \\ 197 \end{bmatrix}$$

To decrypt the cipherteyt $(\mathbf{u}, \mathbf{c})$ we simply need to compute:

$$\begin{aligned} \mathbf{v} = f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{u}) =& f^{-1}(\mathbf{P}^T\mathbf{a} + f(\mathbf{v}) - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ =& f^{-1}((\mathbf{AS} + \mathbf{E})^T\mathbf{a} + f(\mathbf{v} - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ =& f^{-1}(\mathbf{E}^T\mathbf{a} + f(\mathbf{v})). \end{aligned}$$

To avoid decryption errors it's necessary that the entries of $\mathbf{E}^T\mathbf{a}$ are less than $q/(2t)$ in absolute value. To bound these values we need to choose a proper distribution of the entries of the matrix $\mathbf{E}$. We can now decrypt our ciphertext:

$$\mathbf{v} = \left[\frac{5}{255} \left(\begin{bmatrix} 59 \\ 204 \\ 140 \\ 197 \end{bmatrix} - \begin{bmatrix} 211 & 13 & 112 & 111 \\ 192 & 36 & 105 & 173 \\ 83 & 100 & 232 & 20 \\ 37 & 60 & 39 & 225 \\ 65 & 53 & 68 & 159 \end{bmatrix}^T \times \begin{bmatrix} 29 \\ 136 \\ 129 \\ 0 \\ 41 \end{bmatrix} \right) \right] \mod 5$$

$$\mathbf{v} = \begin{bmatrix} 2 \\ 0 \\ 3 \\ 1 \end{bmatrix}$$

We recovered the initial vector message. This is so far the simpler crypto-system with several possible improvements on both a mathematical and algorithmic point of view.

Chapter 5

LWE Variants

The Learning with Errors problem is, so far, one of the most promising lattice based framework in which is possible to build a Public Key crypto-system.

The decision version of the problem, described in a matricial form is: given on input a pair $(\mathbf{A}, \mathbf{v}) \in (\mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m)$ where $\mathbf{v}$ can either be chosen uniformly at random or to be equal to $\mathbf{As} + \mathbf{e}$. Where $\mathbf{s} \in \mathbb{Z}_q^n$ is uniformly chosen at random and $\mathbf{e} \in \mathbb{Z}_q^m$ is chosen according to a certain distribution χ^m that is commonly taken to be a “discrete” Gaussian distribution[1] [18].

The goal is to distinguish when $\mathbf{v}$ is randomly chosen or is equal to $\mathbf{As} + \mathbf{e}$.

This problem enjoy a really tight security reduction to the worst-case lattice problems as approximate-SVP or approximate-SIVP, as already highlighted. And the LWE problem is generally believed to be hard, according to the known attacks to the lattices problems and to the connection to decoding problems.

5.1 Crypto-system Variants

Since the LWE appeared in the literature different variants have been proposed. The idea behind the different systems is the same but the first proposals were all impractical because of the big keys size - think about the keys as full rank matrices $n \times m$ in which all the entries have a size of q, that means which a secure choice of parameters the size of the keys is in the order of MegaBytes. Never the less it is worth to mention the first proposal, Regev’s crypto-system, to have an overview and also because all the other crypto-systems follow from this one.

[1]The tails have not to be cut in an abrupt way, otherwise this could induce a security issue! The statistical distance between the real Gaussian and the discrete one is supposed to be close to 2^{-90} [13]

5.1.1 Regev's crypto-system

The **parameters** of this crypto-system are the integers n, m, l, t, r, q and a real $\alpha > 0$.

The **private key** is $\mathbf{S} \in \mathbb{Z}_q^{n \times l}$ uniformly random chosen, so with high probability full rank.

The **public key** is built starting from two matrices: $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ again uniformly random chosen and $\mathbf{E} \in \mathbb{Z}_q^{m \times l}$ chosen according to a given distribution Ψ_α. The public key is: $(\mathbf{A}, \mathbf{P} = \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times l}$.

The **encryption** process take as input a message from the message space $\mathbf{v} \in \mathbb{Z}_t^l$, then apply to it an encoding f defined as $f : \mathbb{Z}_t^l \to \mathbb{Z}_q^l$ and generate an uniformly random vector $\mathbf{a} \in \{-r, -r+1, \dots, r\}^m$ and output the ciphertext $(\mathbf{u} = \mathbf{A}^T\mathbf{a}, \mathbf{c} = \mathbf{P}^T\mathbf{a} + f(\mathbf{v})) \in (\mathbb{Z}_q^n \times \mathbb{Z}_q^l)$.

The **decryption** take as input a ciphertext $(\mathbf{u}, \mathbf{c})$ and the private key $\mathbf{S}$ and output $f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{v})$.

The decryption will be successful only if the noise added to the encoded message is below a certain threshold (this is related to the encoder). With this decoder the threshold is $q/(2t)$. It is easy to understand that it is better, with this kind of decoder, to encode and encrypt single bits. In this way t will be 2 (that is the minimum) and the overall threshold is therefore $q/4$.

It's evident that this crypto-system has huge keys. But still maintaining this matricial formulation few enhancement can be already performed. An idea is to reduce all the matrices in HNF form (idea from Micciancio), and if we have access to a trusted randomness source we could avoid to store the random $\mathbf{A}$ in the public key and just use an $\mathbf{A}$ drawn from a pre-shared set.

5.1.2 Enhanced Regev's crypto-system

The best enhancement, still remaining in the matrix framework, can be performed considering the operation $\mathbf{AS} + \mathbf{E}$, used in the public key generation, as a reduction modulo a public basis. This reduction can also be done by an attacker and is proven to made the attack harder! In this way we can shrink the public key at essentially no cost.

The enhanced PKC looks like this: the **private key** is defined as: $\mathbf{R}_2$ chosen according to a discrete Gaussian distribution $D_{\mathbb{Z}, s_k}^{n_2 \times l}$ where s_k is the standard deviation.

The **public key** like the previous crypto-system is made of two pieces: a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n_1 \times n_2}$ and a matrix $\mathbf{P}$ defined as: $\mathbf{P} = \mathbf{R}_1 - \mathbf{A} \cdot \mathbf{R}_2$ where $\mathbf{R}_2$ is the private key and $\mathbf{R}_1$ is drawn again from the error distribution, i.e. $\mathbf{R}_1 \in D_{\mathbb{Z}, s_k}^{n_1 \times l}$.

The **encryption** takes as input $(\mathbf{A}, \mathbf{P}, \mathbf{m} \in \Sigma^l)$ where $\mathbf{m} \in \Sigma^l$ is the message taken from the message space. Generate the error vectors: $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3) \in \mathbb{Z}^{n_1} \times \mathbb{Z}^{n_2} \times \mathbb{Z}^l$ with each entry drawn independently from $D_{\mathbb{Z},s_k}$. Prior to the encryption encode the message, so $\bar{\mathbf{m}} = encode(\mathbf{m}) \in \mathbb{Z}_q^l$.

The ciphertext is made of two pieces: $(\mathbf{c}_1^t, \mathbf{c}_2^t) = (\mathbf{e}_1^t\mathbf{A} + \mathbf{e}_2^t, \mathbf{e}_1^t\mathbf{P} + \mathbf{e}_3^t + \bar{\mathbf{m}})$.

Even though the polynomial e_3 looks useless it increase the security of the crypto-system again a chosen plaintext attack[2].

The **decryption** algorithm takes as input the ciphertext and output: $decode(\mathbf{c}_1^t \cdot \mathbf{R}_2 + \mathbf{c}_2^t)^t \in \Sigma^l$. Substituting the ciphertext in the equation have the following result:

$$\mathbf{c}_1^t \cdot \mathbf{R}_2 + \mathbf{c}_2^t = \mathbf{e}_1^t\mathbf{R}_1 + \mathbf{e}_2^t\mathbf{R}_2 + \mathbf{e}_3^t + \bar{m}^t \tag{5.1}$$

To recover the original message we need to apply the decoder to this perturbed and encoded message.

It is clear that for the decryption to be successful the error added to the message has to be under a certain threshold that is related to the encoder used. In the case of the encoder used in the classical version of the LWE, we simply multiply the bits times a constant value. This constant value is taken as the half value of the modulo of the ring: i.e. $q/2$.

The security of the crypto-system is related to the ratio q/s_k, where s_k is the standard deviation of the distribution of the error. The problem becomes easier for larger ratios! So it could be a good idea to use a different encoding mechanism that has an higher threshold and therefore allow to use a bigger standard deviation or a smaller modulo q.

5.1.3 LWE Ring Variant

Following the example of the NTRU crypto-system the LWE crypto-system can be built in a Ring modulo a cyclotomic polynomial, i.e. $\mathbb{Z}_q/\phi_m$.

This allows to shrink the keys and to perform several operations in a faster way. Like the polynomial multiplication with the DFT [27].

In doing so we face few issues. For example is not straightforward to extend the security reduction from the LWE to the Ring-LWE. But so far we don't known any algorithm that can exploit the structure of the ideal lattice to perform a better attack.

[2]Chosen plaintext attack is a theoretical attack model in which an attacker has access to an oracle that encrypt a polynomial number of messages chosen from the attacker himself. This is a situation that happen very likely since in a PKI the public key and the algorithm is public.

Another problem is that the errors are sampled from a rounded spherical Gaussian distribution in the field of integers but if you sample them in the ring than it is not so clear what is the shape of the Gaussian. It would be better to sample in the field and then transform the sampled values back to the ring, to guarantee a theoretical security [39]. This is of course source of inefficiency!

The crypto-system is similar to the enhanced version of Regev's one but all the operations take place in the ring: $\mathbb{Z}_q[X]/(X^n+1)$, with n power of 2, that define the anti-cyclic lattices.

The keys are two polynomials. To speed up all the operations we could perform the NNT transformation on all of them in order to have the multiplication component-wise. A more general way of performing polynomial multiplication in a ring modulo x^m+1 is to to embed the ring in a bigger ring modulo x^N-1 where (x^m+1) divide (x^N-1). The embedding can be done padding zeros to the original vector/polynomial. In the ring modulo (x^N-1) all the multiplications can be carried out with the Discrete Fourier Transform (DFT). This is a "waste" of operations compared to the NNT but it is more general and the anti-cyclic ring doesn't need to have an exponent of the type 2^m but can be any number without special properties.

The **private key** is a polynomial r_2 with coefficients chosen according to a discrete Gaussian distribution. In few proposals [13, 60, 54] the polynomial r_2 is chosen in a a distribution of $0,1^n$. This would make the key generation faster but actually it is not crucial since a key pair is generated not so often.

The **public key** is generated choosing two more polynomials, a from a random distribution and r_1 from a Gaussian one. From this pieces we evaluate the polynomial $p = r_1 - a \cdot r_2$ in the ring. The public key will therefore be: (a, p).

The **encryption** take as input the public key and the message and generate three error polynomials e_1, e_2, e_3 sampled from the Gaussian distribution. After encoding the message it can be encrypted like the previous PKC:

$$(c_1, c_2) = (a \cdot e_1 + e_2, p \cdot e_1 + e_3 + \bar{m}) \tag{5.2}$$

The **decryption** is in the same spirit of the previous PKC too. It takes as input the ciphertext and the private key and output the message after decoding it: $m = decode(c_1 \cdot r_2 + c_2)$.

All this operation can be performed using a sort of DFT but in order to do so we need to apply this transformation to every and each polynomial, but the way the crypto-system works is the same.

Chapter 6

Personal Implementation of the ring-LWE PKC

In this last chapter I will show the results of the investigations that I conducted on the ring-LWE crypto-system. I focused my efforts on a practical implementation of an encryption algorithm, based on the ring-LWE, individuating two points that could be eventually enhanced.

I implemented the algorithm using Sage [62], a free open-source mathematics software system that builds on top of many existing open-source packages accessible through a common, Python-based language. In the next paragraphs I will briefly summarize the ring-LWE details highlighting the ones that I will enhance in my implementation.

The strength of ring-LWE PKC is its extremely simple operations behind the various cryptographic tasks:

- the **key generation**: consists of picking up two polynomials (r_1, r_2), form a Gaussian distribution, and picking up a polynomial a, from a uniform distribution in the ring. The polynomial r_2 will be used as private key, on the other side, the public key p, will result from the operation: $p = r_1 - a \cdot r_2$ performed in the modular ring.

- the **encryption**: consists of picking up three polynomials (e_1, e_2, e_3), from a Gaussian distribution, and performing the operations: $(a \cdot e_1 + e_2, p \cdot e_1 + e_3 + \bar{m})$. These are respectively two parts of the ciphertext.

- the **decryption** depends on the operation: $c_1 \cdot r_2 + c_2$

On top of the encryption/decryption it is defined an encoder/decoder. In most cases, and also in my implementation, it is a simple function mapping the space $\{0, 1\}^n \to \{0, q\}^n$

just multiplying a bit by $\lfloor q/2 \rceil$. This will result in a 0 or in a $\lfloor q/2 \rceil$. The decoding is just the inverse of this map: resulting in a 0 decoding for values less or equal than $\lfloor q/2 \rceil$ and so on.

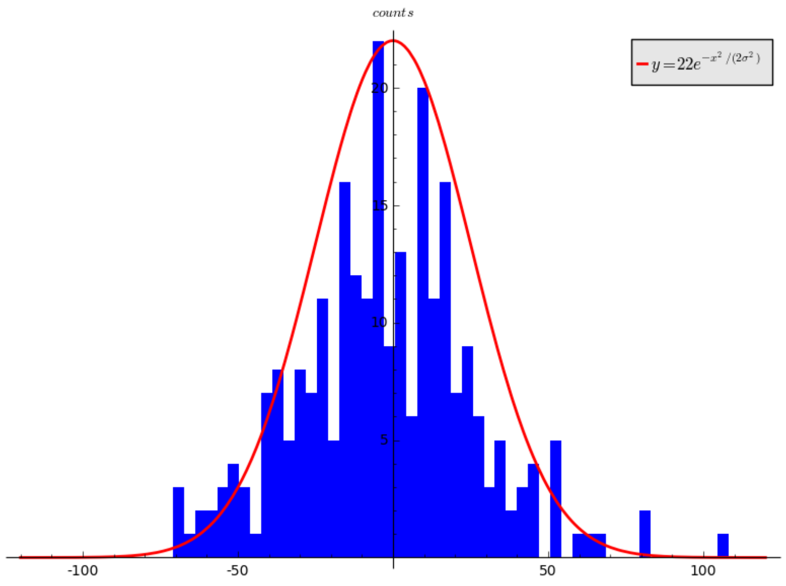
It is easy to understand that the operations involved are only the *multiplication, in the modular ring*, and the *sampling* procedure. In the key generation there is both a sampling from a *uniform* distribution and from the *Gaussian* distribution. In the encryption there a three sampling from the *Gaussian* distribution. Finally the decryption is made of only one multiplication in the modular ring.

The addiction are not taken into account because they are the fastest operation possible, to give an example with sage I can perform the multiplication between two polynomials with $n = 256$ coefficients modulo $q = 6871$ in the ring modulo $x^{256} + 1$ in time $t \simeq 600\mu s$ against the sum time $t \simeq 90\mu s$. The sum takes almost one tenth of the multiplication. With ad hoc implementations the multiplication time can drop to less than $100\mu s$ being comparable with the sum time.

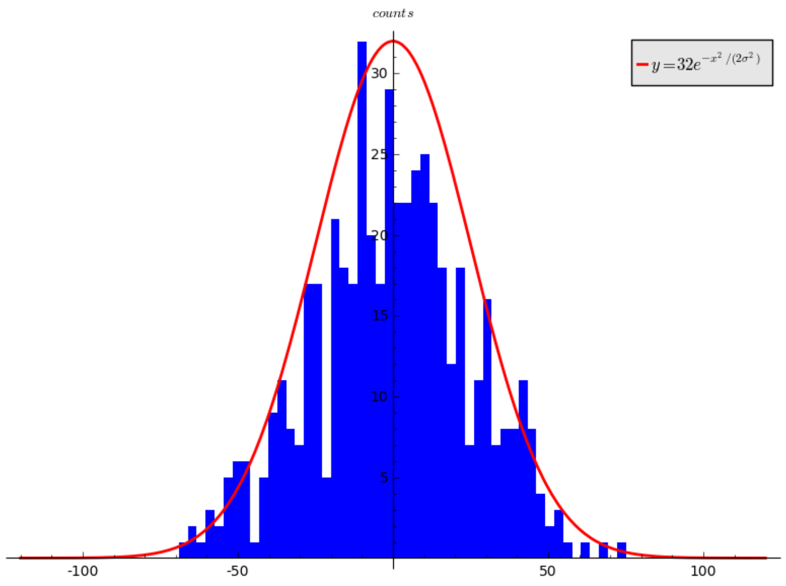
So far the biggest source of inefficiency comes from the sampling methods. Sage implements the sampling via the rejection sampling, also known as acceptance-rejection method and it is a type of Monte Carlo method. Basically this technique takes sample from a uniform distribution, with a density function bigger than the desired distribution, and rejects all the samples that are not in accordance with the target distribution.

Since the R-LWE implementation is at a ”toy model” level, sage can be seen like a black box without demanding for a complete understanding of all the algorithms behind. Therefore, for instance, we could employ other techniques for the sampling to improve the implementation. There is plenty of them in the literature, like the fast but memory consuming *Knuth-Yao* algorithm, that is based on a random walk on a binary tree with the Gaussian samples in the leaves, or others algorithms with other trade-offs.

At this stage I employed for the sampling built-in sage methods, all based on the rejection-sampling technique, even if the quality could be poor in some cases. See for example the distribution of the samples chosen according to a Gaussian distribution in the picture6.1. The quality of the samples is the best obtainable with the built-in sampler of sage even if it appears pretty unbalanced compared with the red shape that is the ”theoretical Gaussian”. This could be partially due to the small numbers of samples or to the abrupt cut of the tails that is by default at 6σ (in the literature usually it is close to 12σ). The situation get worst with the uniform samples. As it can be seen in the picture 6.2 the samples are gathered around few numbers. The situation looks more uniform if

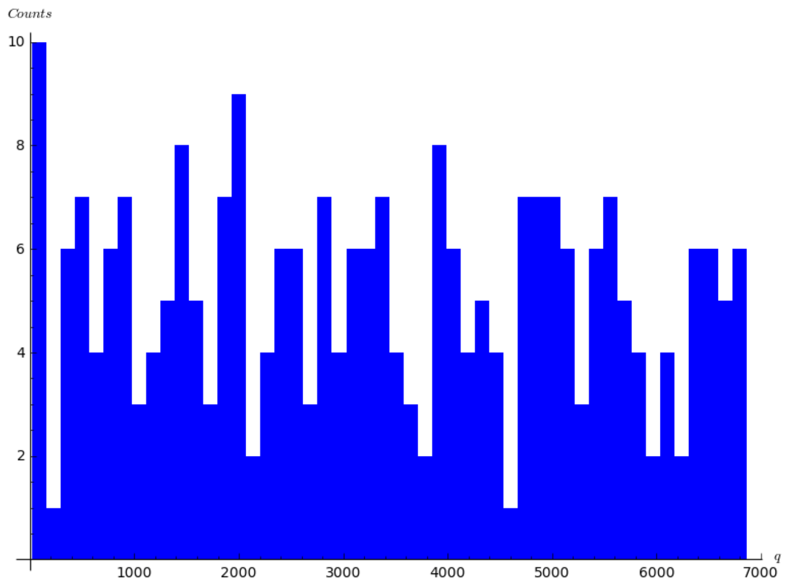


(a) *256 samples chosen according to a Gaussian distribution with* $\sigma = 25$.

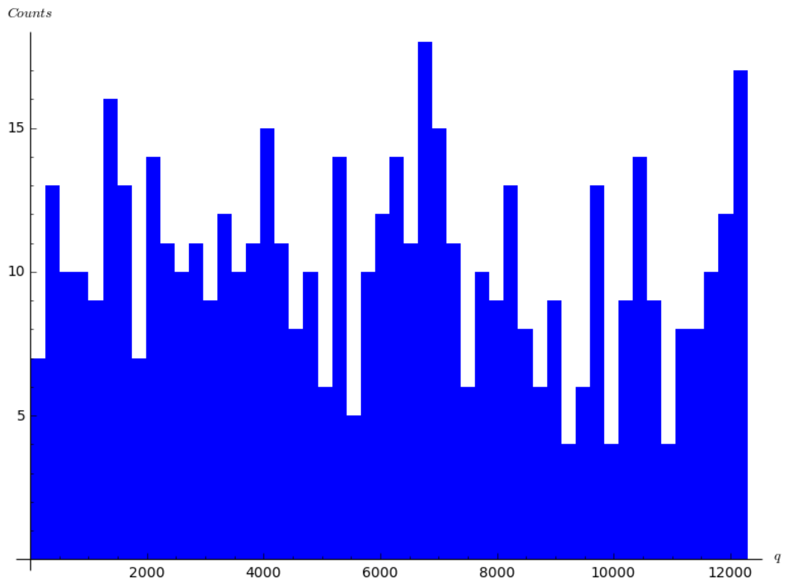


(b) *512 samples chosen according to a Gaussian distribution with* $\sigma = 25$.

Figure 6.1: Two examples of Gaussian distribution samples.



(a) *256 samples chosen according to a uniform distribution with* $\sigma = 25$.



(b) *512 samples chosen according to a uniform distribution with* $\sigma = 25$.

Figure 6.2: Two examples of uniform distribution samples.

we increase the samples. For a practical implementation it is mandatory to have better sampler.

In my implementation I considered only two set of parameters (n, q, σ): $(256, 6871, \sigma)$ and $(512, 12289, \sigma)$ where $\sigma \simeq 12$. This is why in the previous pictures the distributions have that particular number of samples and the width of the histograms is q. This two set of parameters according to the literature are at a 128 and 256 security bits level. A security bit level can roughly be seen as an estimate of the security level: it is the number of operations needed to recover the message or the key in a brute force attack, so 128 bits security level means that in order to recover the plaintext we need to perform $2^{128} \sim 10^{38}$ operations.

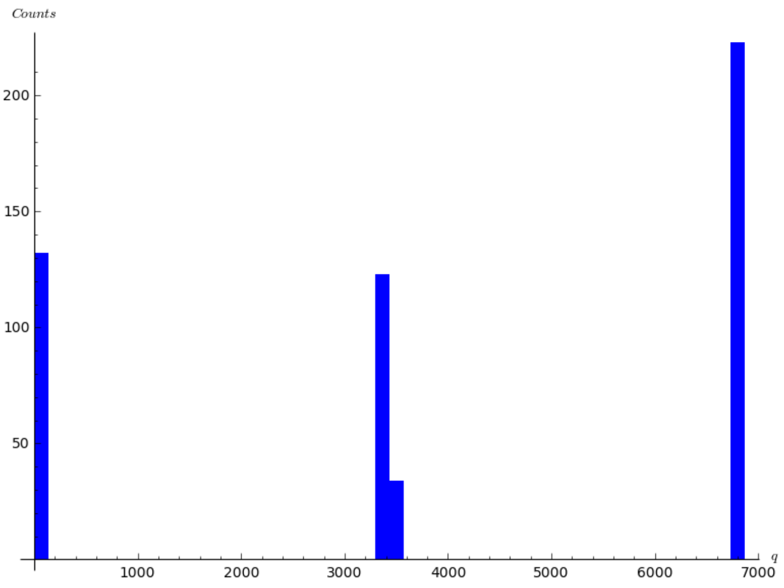
A good estimate of the security of the PKC, or in other words: the difficulty of solving the LWE problem upon which is based both the security of the messages and the keys, is the ratio q/σ. Higher the ratio, easier the problem: because the Gaussian noise added is narrower.

The entire crypto-system could be described in physical words like a "conversion" of digital signal to an analogical and slightly perturbed one, for what concern the encryption. The decryption would be the inverse of this process. Therefor the width of Gaussian over the size of space is a good estimate of the security level.

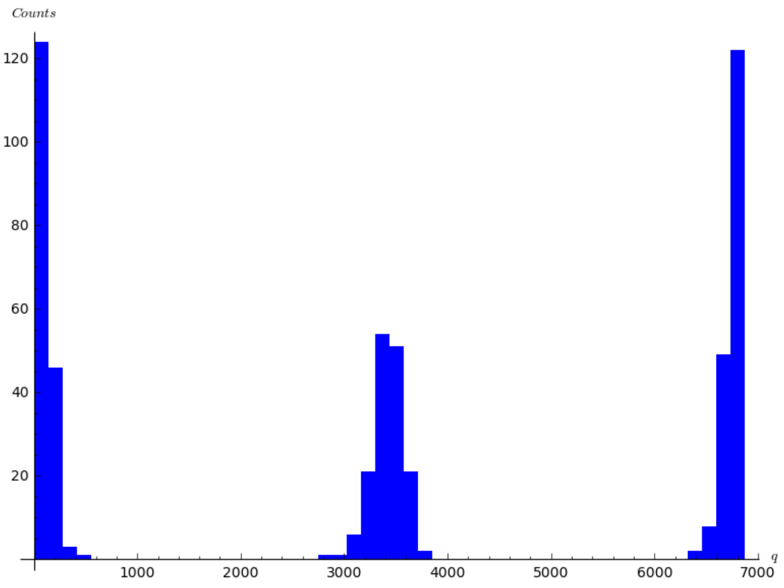
Before going further it would be interesting to analyze the blow-up factor of the ciphertext. The encoding procedure introduce a blow-up due to the mapping from a bit to a value of size q: that means from $1 \to \log_2 q$. Furthermore the ciphertext is made up of two components (c_1, c_2) of length n. So without taking into account a possible padding we increase the size from n bits $\to 2n \log_2 q$ bits. A 256 bits plaintext file become a 6K bit ciphertext circa, with $q = 6871$ and 7 Kbits for $q = 12289$. The size increase is high but not of the order of few Megabits like the standard LWE.

6.1 Enhancements/Modifications to the original Ring-LWE

The first modification of the system is due to the faulty behavior of the embedded algorithm responsible for the generation of samples distributed according to a Gaussian distribution. In the following it will be simply referred as Gaussian samplers. The high unbalance of the samplers make the threshold decoder proposed in the original construction ineffective leading to completely wrong decoding already to the smaller values of σ, i.e. $\sigma \simeq 0.4$. In the picture 6.3 we can already realize how ineffective the $q/2$ threshold decoder would be already at a really low σ. The $q/2$ would decode as 1 all the values over $q/2$ and as 0 all



(a) *Result of the decryption with* $\sigma = 0.4$.



(b) *Result of the decryption with* $\sigma = 12$.

Figure 6.3: Two decrypted messages before the application of the decoder.

the others below $q/2$.

Nevertheless it is not all lost because the unbalanced Gaussian perturbation is nothing more than shift of the coefficient in our ring of a fixed value and all the coefficient of the decrypted polynomial that correspond to a 1 are displace around the value of $q/2$ with a half-width of $\pm q/4$. Therefore I defined a decoder that simply map all the values included between the boundaries $q/2 - q/4$ and $q/2 + q/4$ to 1 and the remaining one to 0. As it can be seen from the picture 6.3 even a value of $\sigma \simeq 12$ allows a complete decode[1] due to the narrow displacement of the polynomial coefficients around $q/2$. The decoder behave as expected also for the set of parameters $(n = 512, q = 12289, \sigma = 12)$.

Apart from the decoder adaptation and keeping in mind the "issues" of the R-LWE I came out with two possible enhancements.

For sake of completeness I have to say that in the literature there is already a small enhancement related to the key generation. To recall, the private key is r_2 and the public key is a *perturbed product* of the form: $r_1 - a \cdot r_2$. Where both r_1 and r_2 are sampled from a Gaussian distribution. The idea is to sample r_2 from a uniform distribution in $\{0, 1\}^n$ instead of the original one. This is a much more faster operation, although it is a minor enhancement since the key generation take place a limited number of times (ideally each user generate a key pair once every three years according to the protocols used today, unless the keys are somehow compromised). It could be necessary to make the key generation faster and "lighter" if we think that it could be accomplished by a small cryptoprocessor like the crypto-chip on a smart-card. This is really likely to be the case according to today best-practices.

The security of the private key, i.e the computational difficulty of recovering the private key from the public one, still holds and it follows from the *subset sum problem* problem an *NP-complete* problem. It is mandatory now to clarify that the standard construction of the LWE PKC enjoy the security of the Learning With Errors problem, but it holds only if the private key is sampled from a Gaussian distribution. So we can assume that this variant with the private key sampled from a uniform distribution bounded between zero and one even if doesn't enjoy the security of the LWE problem it is close related to the subset sum problem and can be reduced to it[2].

[1]The complete decoding is possible even with large σ due to the fact that in the encryption the polynomial e_1 is sampled from a uniform distribution $\in 0, 1^n$, but if e_1 is drawn from a Gaussian distribution the recover of the original message is far more difficult also because of the high unbalance of my sampler algorithms.

[2]I would like to remind once more that the ring-LWE variant, that is analyzed here, doesn't enjoy any theoretical security reduction to these problems but so far there is no algorithm capable of exploit the

$function$	$distribution$	$hamming - encoding$	$time$
$key - gen$	$uniform$		$46 \pm 1ms$
	$Gaussian$		$49 \pm 1ms$
$encryption$	$uniform$	NO	$138 \pm 1ms$
	$Gaussian$	NO	$140 \pm 1ms$
	$uniform$	YES	$145 \pm 1ms$
	$Gaussian$	YES	$147 \pm 1ms$
$decryption$	$uniform$	NO	$140 \pm 1ms$
	$Gaussian$	NO	$141 \pm 1ms$
	$uniform$	YES	$188 \pm 1ms$
	$Gaussian$	YES	$190 \pm 1ms$

Table 6.1: Execution time of the main crypto-function with $(n, q, \sigma) = (256, 6871, 12)$

The *subset sum problem* is defined as: given a set of numbers, in our case vectors, is there a non-empty subset whose sum is zero? (we can apply an offset to the zero and define a target sum). Coming back to the definition of the public key $p = r_1 - a \cdot r_2$, we can see the vector as the "set" of numbers and $-p + r_1$ as the target vector, so the problem is to find a combination of as that sum to the target vector: this is exactly the same that recover the private key r_2. The problem get harder if r_2 has a short norm, i.e. a lot of zeroes. If r_2 has a small number or not at all zeroes it would be easy to recover it via Gaussian operation on the non-homogeneous, full rank system defined by n equations of the type: $\sum_{x=0}^{n-1} r_{2,x} \cdot \mathbf{a} = -p_x + r_{1,x}$.

Keeping in mind this enhancement I propose to apply it also to the encryption procedure and sampling the coefficients of the polynomial e_1 from a $\{0,1\}^n$ distribution instead of a Gaussian one. Recalling the shape of the ciphertext: $(c_1, c_2) = (a \cdot e_1 + e_2, p \cdot e_1 + e_3 + \bar{m})$, it is clear that c_1 and c_2 have the very same structure of the public key. Therefore they enjoy the same security reduction to the subset sum problem and actually e_1 with a lot of zeros would make the PKC more secure and faster at the same time, because the sampling of the e_1 coefficients have to be done each and every time a segment of the plaintext is

more "structured" construction of the ring-LWE and it seems that this situation will hold longer.

$function$	$distribution$	$hamming - encoding$	$time$
$key - gen$	$uniform$		$60 \pm 1ms$
	$Gaussian$		$58 \pm 1ms$
$encryption$	$uniform$	NO	$162 \pm 1ms$
	$Gaussian$	NO	$160 \pm 1ms$
	$uniform$	YES	$178 \pm 1ms$
	$Gaussian$	YES	$175 \pm 1ms$
$decryption$	$uniform$	NO	$161 \pm 1ms$
	$Gaussian$	NO	$162 \pm 1ms$
	$uniform$	YES	$235 \pm 1ms$
	$Gaussian$	YES	$235 \pm 1ms$

Table 6.2: Execution time of the main crypto-function with $(n, q, \sigma) = (512, 12289, 12)$

encrypted.

See the tables 6.1 and 6.2 for time performance comparisons between the sampling from a Gaussian distribution and a uniform one bounded between 0 and 1. All the values are referred to the encryption of a message of 256 bits. This is the standard size of a symmetric key and usually in the encryption process the asymmetric crypto-systems are employed to exchange the key of a symmetric crypto-system because exchanging standard (and usually bigger) messages would be too expensive. This is referred as hybrid crypto-system. From the tables it is evident that for an high degree polynomial the Gaussian sampling is far more efficient than the uniform one but this is only due to the implementation since the theoretical algorithmic complexity is lower in a uniform sampler. The encryption using the uniform sampler becomes more convenient with larger messages and, more important, as it will be shown later it allows to use larger σ in the whole encryption process.

With the first proposal I had in mind to enhance the time related performances, but what about the correctness of the decryption? Can we make the decryption problem harder for an attacker without decreasing the correctness of the decryption? The ratio q/σ is a rough estimate of the security, higher the ratio lower the security. To increase it we could for example increase the value of the standard deviation σ of the Gaussian sampler. This way the probability of recover the message decrease but there is something that can increase the probability of recover the correct message.

The idea is to embed a code-correcting algorithm prior to the standard algorithm encoding: in my implementation I chose the Hamming correcting code. This idea is basically to add 3 parity bits every 4 bits. This code is able to correct up to 1 error and to detect 2 errors without correcting them. With more than 2 errors the code is useless.

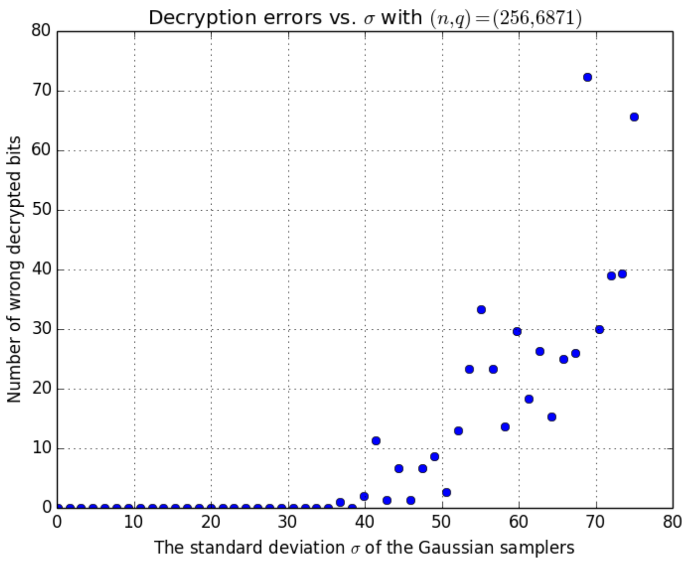
This enhancement, on the other side, comes at a great cost: the blow-up factor of the ciphertext gains another factor 2 circa, because we now need 7 bits to encode only 4 bits. The blow-up ratio become roughly $4\log_2(q)$.

The introduction of the hamming correcting code, in the implementation the polynomial e_1 is sampled from the uniform distribution, is not effective enough to justify the huge blow-up factor. From the picture[3] 6.4 it is possible to see that with the hamming code we can push the σ to a value of over 40 with the encoding activated, against a value of slightly more than 35 with no encoding. We can therefore use a 14% larger σ but with a almost doubled message size. The situation is a bit better with the parameters $(512, 12289, \sigma)$, see the picture 6.5, it is possible to have 23% larger σ.

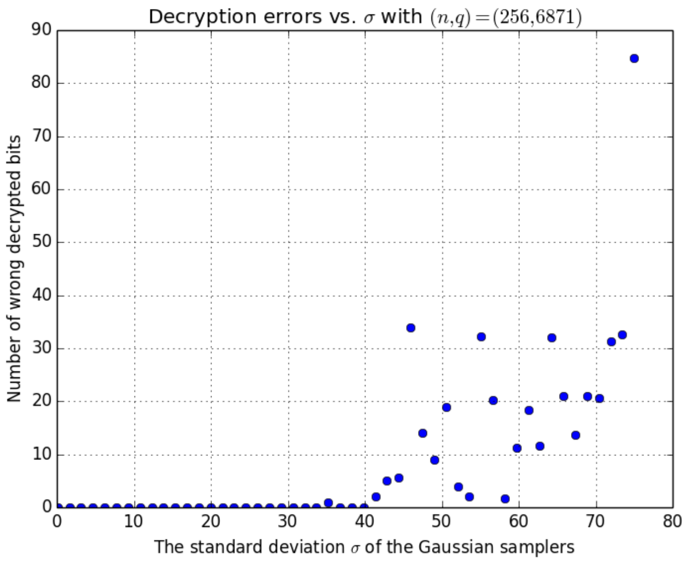
So far the situation looks great, with my construction I can reach values of σ almost four times larger than the one used in the literature and it seems a really big enhancement of security and, considered that my Gaussian samplers are highly unbalanced, it will be possible to reach higher values of σ employing higher quality samplers. On the other side the situation completely changes if also the polynomial e_1 is sampled from a Gaussian distribution. From the picture 6.6 it is possible to see that due to the low quality of the sampler σ is slightly lower than literature, i.e 12. In this situation the hamming encoding allows to reach a 20% higher σ: actually it seems that the Hamming encoding allows to reach 20% circa higher σ regardless the parameters of the crypto-system. At this stage it is not clear if sampling e_1 from the uniform distribution makes the crypto-system weaker. Since it is the polynomial multiplied with the public key it shouldn't compromise the security. On the other side the other two error polynomial are just added and they have to be Gaussian-like in order to give to the ciphertext a random shape and guaranty the semantic security of the crypto-system.

Although sampling from the uniform distribution could be seen as a vulnerability it allows to use really larger σ for the other polynomial sampled, i.e.

[3]The pictures of errors vs. σ are obtained executing three encryption/decryption cycle per point for fifty values of σ.

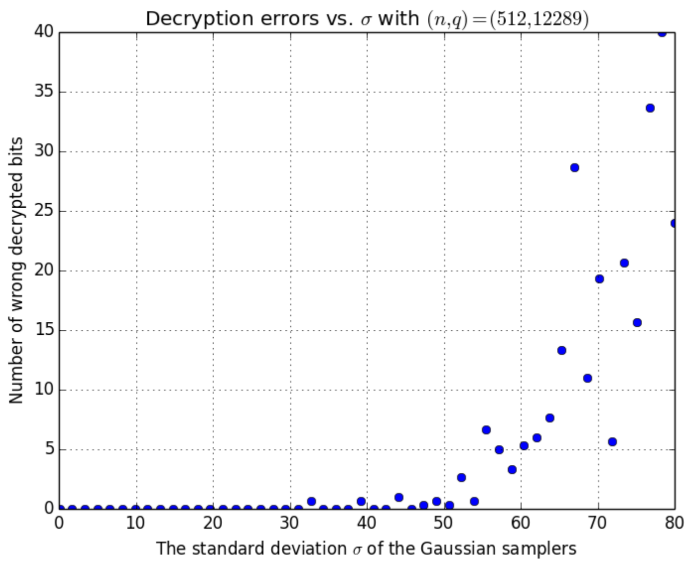


(a) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1 \in \{0,1\}^n$ and the Hamming encoding OFF.*

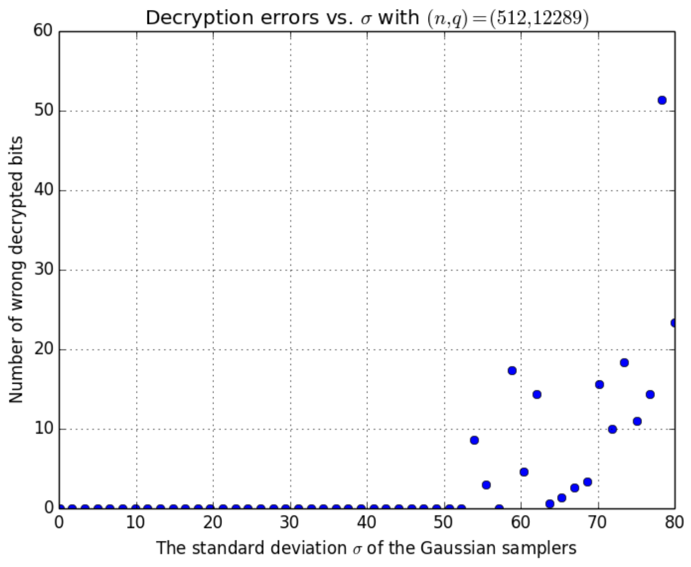


(b) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1 \in \{0,1\}^n$ and the Hamming encoding ON.*

Figure 6.4: Comparison of the decryption errors for Hamming encoding ON/OFF.

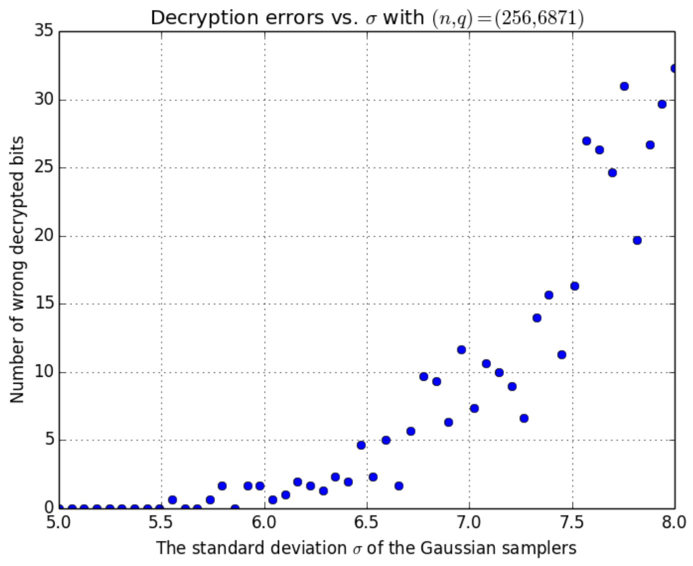


(a) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1 \in \{0,1\}^n$ and the Hamming encoding OFF.*

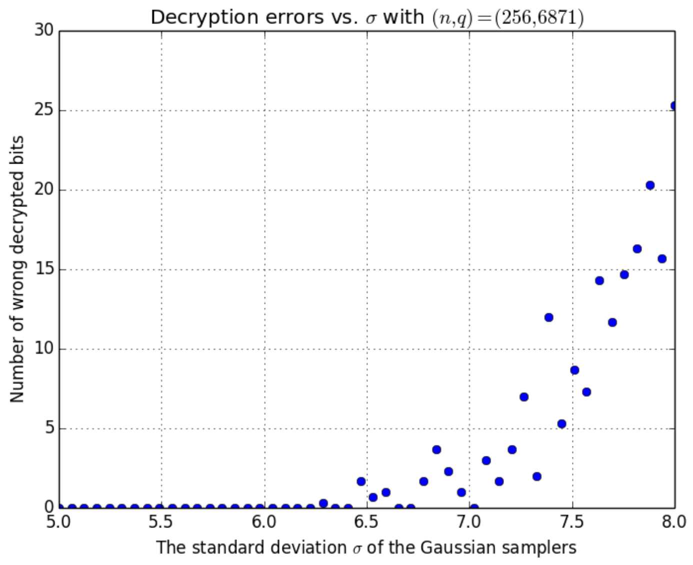


(b) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1 \in \{0,1\}^n$ and the Hamming encoding ON.*

Figure 6.5: Comparison of the decryption errors for Hamming encoding ON/OFF.



(a) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1$ sampled in Gaussian distribution and the Hamming encoding OFF.*



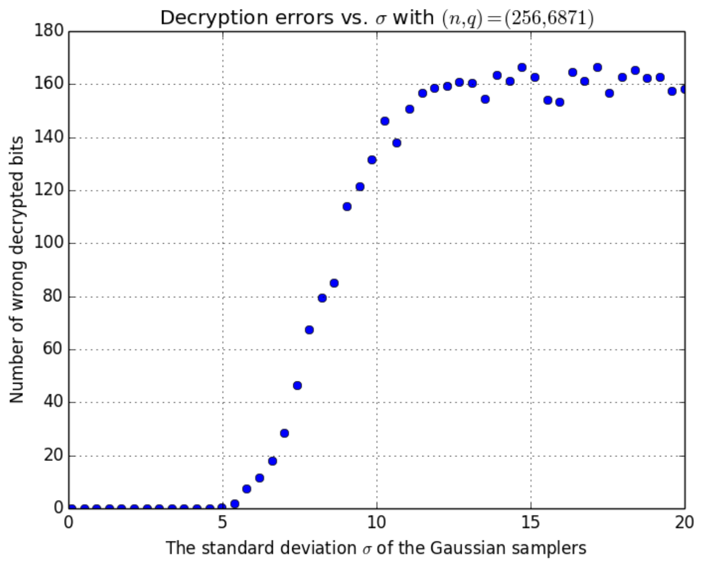
(b) *Wrong decrypted bits vs. σ with $r_2 \in \{0,1\}^n, e_1$ sampled in Gaussian distribution and the Hamming encoding ON.*

Figure 6.6: Comparison of the decryption errors for Hamming encoding ON/OFF with e_1 sampled from a Gaussian distribution.

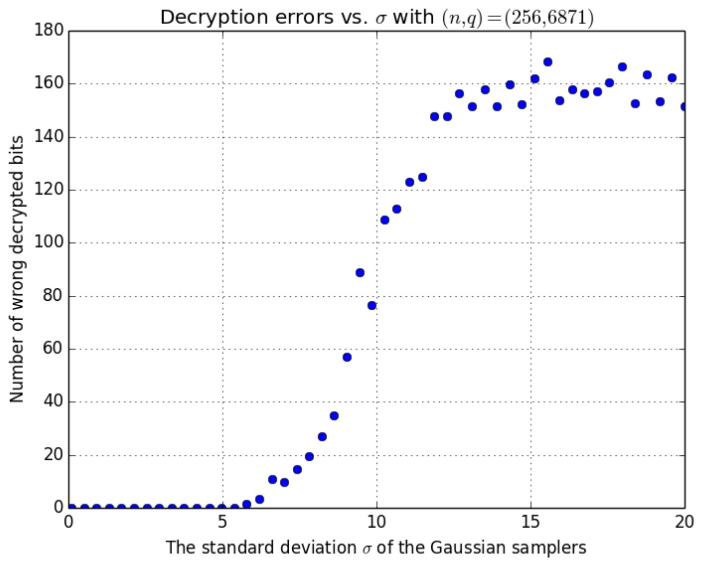
- r_1: used in the key generation to add noise to the product between the private key polynomial and a random matrix. Actually this random matrix in few articles is been proposed to be public, to reduce the size of the keys and make the key exchange easier, and therefore a ”more” random noise makes the LWE problem, related to the key recovery, harder.
- e_2: used in the first component of the ciphertext. It has the same function that r_1 has in the key-pair generation.
- e_3: used to add randomness to the second component of the ciphertext and therefore used to protect the message from CPA attacks.

So it is not clear if this modification afflicts the security or increase it making the encryption process even faster. Further studies are needed.

It is worth also to mention that sampling the private-key polynomial r_2 form a Gaussian distribution highly increase the probability of a faulty decryption even if e_1 is sampled from the uniform distribution and prevent us from using values of σ for the other samplers higher than 6, at least in my implementation. On the other side sampling both r_2 and e_1 from a Gaussian distribution doesn't decrease the threshold of σ for a correct decryption: see picture 6.7. From this pictures it is clear that it is a good idea to sample r_2 from a uniform distribution as long as also e_1 is sampled from the same distribution, otherwise it brings no benefits in terms of correctness and probably make the security lower but the key generation a bit faster.



(a) *Wrong decrypted bits vs. σ with r_2 and e_1 sampled in Gaussian distribution and the Hamming encoding OFF.*



(b) *Wrong decrypted bits vs. σ with r_2 sampled in Gaussian distribution and $e_1 \in \{0,1\}^n$ and the Hamming encoding OFF.*

Figure 6.7: Comparison of the decryption errors with e_1 sampled from the uniform distribution first and a Gaussian after, r_2 sampled from a Gaussian one.

Chapter 7

Conclusions

When I started my work on the *Post Quantum Cryptography* I was asked by the Airbus Group and in particular by professor Helmut Kaufmann, working in the Airbus Innovation, to give him a high level overview on the possible alternatives of the current stave of the art encryption algorithms, like the RSA. And possibly to recommend a candidate replacement for a future utilization in the corporation.

The main difficulties that I faced during my studies came from the nature of the post-quantum cryptography itself, whose security is hardly based on strong mathematical proofs but is related to the difficulty of algorithmic problems somehow related to the crypto-systems. For example solving a system of undetermined random equations is a difficult task but if we wish to hide a message in a set of equations we have to make them solvable somehow, in other words we need to hide a trapdoor that makes the problem easy to solve for somebody that has a key and it is supposed to keep the problem difficult for all the other people. Hiding a trapdoor means hiding a pattern and translating it back to the system of equations: they cannot be random anymore and therefore they have a pattern. So it is easy to understand that the security of this system comes from the hardness of solving a random set of undetermined equations but it is a reduced version and it is considered secure until somebody will find and exploit and will break the system.

This example makes clear the first difficulty I faced: cryptography is "considered" secure until a more efficient way of solving the underlying problem is found. This, on the other side, makes cryptography an interesting and fast moving field.

All the crypto-systems I studied so far have some good qualities and some are bad and this made really difficult for me to choose a direction. In the end I chose to investigate the lattice based crypto-systems for all the reasons that I explained before but still between these algorithms choosing one is still difficult. Very briefly, in this family there are two main alternatives, the NTRU one, build in a cyclic lattice environment whose hardness is not based on any algorithmic problem, but completely empiric, and the ring-LWE, built in an anti-cyclic lattice environment whose hardness is based on the famous learning with errors problem.

A good rule of thumb in cryptography is to choose the algorithm that has the most trust from the community, and using this basic parameter the only algorithm that I could recommend is the NTRU one. It was implemented almost a decade ago and from then it is continuously under cryptoanalitic investigation. Furthermore it is patented (IEEE P1363) and already distributed and used by some customers of the company that developed it. These are all good reason to stop my studies and to considered the company goal complete.

Moved by curiosity we also decided to investigate the ring-LWE crypto-system and eventually come up with an implementation. We decided to to this because the system is very interesting from a theoretical point of view, because its hardness is based on a theoretical problem, and also for a possible implementation because it showed its potential with very fast operations. It is also interesting for possible application in the homomorphic field, another fast moving field of the cryptography with possible application in a cloud environment.

At the current stage I implemented a toy model of the crypto-system with some possible enhancement, with all the result shown in the previous chapter. Changing the sampling of the polynomial e_1 produce a really interesting result that make the system faster and stronger even though moving from a Gaussian sampler to a uniform one bounded in $\{0, 1\}$ is a bit abrupt: it could be worthy to analyze the behavior of the system with other distributions "pseudo-Gaussian" and to better analyze the security. On the other side the embedding of the hamming correcting code doesn't give a boost big enough to justify a factor 2 in the blow-up factor ($size(\text{ciphertext}) \simeq 2{\cdot}size(\text{message})$) at least if e_1 is sampled from a uniform distribution. It would be more useful if e_1 were sampled from a Gaussian distribution only in case we need a protocol in which multiple and subsequent encryptions are needed.

Surely the ring-LWE is one of the most promising proposal in the lattice-based crypto-system family but, and this is a big but, it needs to be systematically studied for a longer time in order to gain the trust we need to use it.

To conclude I think that post-quantum cryptography offers a lot of good ideas for new crypto-systems resistant against quantum attacks, that could even be implemented in a short time, compared to the time required to implement other solutions, but also may provide new and better ways of doing ordinary cryptography. One example is the homomorphic cryptography that is fast developing in this period. The only reason why post-quantum crypto-systems are not developed yet, actually not even well known, is that, for the time being, algorithms, like the R.S.A, offers a simpler alternative than the post-quantum proposals but they don't take into account quantum computers attacks, and this could be the boost needed to go further with the research.

Appendix A

Scripts

The Ring LWE cryposystem is being implemented using *sage 6.4.1*, a powerful interface that works with the most known libraries, like the NTL library. The advantage is that all the libraries are written is several computing languages but the sage interface add another level that allows to use only a "Python-style" language to use all the libraries without be forced to know all the details of every and each library.

I used sage in particular to implement the algebra in the ring modulo a polynomial. This is the basic building block of the modern crypto-systems. Even if in most of the literature all the implementations pointed to a fastest execution possible, that means starting from a faster way of performing polynomials multiplications, I implemented my system at a very high level without being concerned about the speed. With sage, and my current PC configuration, I am able to perform a multiplication of two polynomials in the quotient ring $\mathbb{Z}_q/x^n+1$ for $n = 256$ and $q = 6871$ in few hundreds of microseconds against the most performing implementations that can reach the peak of tens of microseconds per multiplication with the same parameters choice.

In the following the three functions that make the crypto-system:

A.1 key-pair generation

```
# -----------------------
# SCRIPT BY ANTONIO VAIRA
# 01.12.2014
# -----------------------
# this script is resposible for the "key generation" of the ring-LWE crypto-system.
# It picks up from a given distribution the coefficients of 3 Polynomials:
```

```
# a, r1, r2 and return the public key (a,p = r1 - ar2) and the private key(r2).
# The polynomials are randomly generated in the polynomial quotient ring
# and all the operation are handled by sage (NTL implementation).

# INPUT
# - n: size of the "lattice" or "degree +1" of the polynomials
# - q: modulus of the GF
# - std_dev: std deviation for the gaussian distribution
# - flag_secret_key: if 1 r2 is generated from {0,1}
#----------
# OUTPUT
# - the private and public keys

# -----modules for the sampling-----  #
from sage.crypto.lwe import UniformPolynomialSampler
from sage.stats.distributions.discrete_gaussian_polynomial
import DiscreteGaussianDistributionPolynomialSampler

# -----modules quotient ring operations-----
from sage.all import FiniteField
from sage.all import PolynomialRing

def key_gen(n, q, std_dev, flag_secret_key=1):

  # define the polynomial quotient ring Q: perform the coercion to the ring Q
  R = PolynomialRing(FiniteField(q),'x')
  x = R.gen()
  Q = R.quotient(x**n +1, 'x')
  x = Q.gen()

  # generate uniformly random the piece of the public key "a" from Q
  a = Q.random_element()

  # control condition: if flag_secret_key = 1 r2 is a binary polynomial
  # AND r1 is drawn from a gaussian distribution
  # else they are both drawn from a gaussian distribution
  if flag_secret_key:
    print "The secret key is generated from an uniform distribution in {0,1}^n"

    # generate the binary polynomial r2 that will be multyplied by a
    r2 = UniformPolynomialSampler(Q, n, 0, 1)()

    # generate the polynomial r1 from with coefficients drawn
    # from a gaussian distribution
    r1 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()

  else:
    print "The secret key is generated from a gaussian distribution!"

    r1 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()
    r2 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()

  p = r1 - a*r2

  return(a,p,r2)
```

A.2 encrypt plus hamming encoding

```
# -----------------------
# SCRIPT BY ANTONIO VAIRA
# 16.12.2014
# -----------------------

# this FIRST script is responsible for the ascii encoding


# of a certain number of characters and the subsequent encoding
# using an Hamming with minimum distance 3.

# INPUT
# - "string" to encode in ascii and padded with 3 parity bits
#  (Hamming Encoding)
#----------
# OUTPUT
# - the binary representetion of the characters divided in block of
# 4 bits + 3 parity check bits
# the format of the output is a list of integer
# (can be easily converted in a vector)

from sage.crypto.util import ascii_to_bin

def ascii_hamming_encode(n, flag):

  # The idea is to define a dictionary to map the half of the
  # binary representetion to the codewords!

  # The flag (1 by default) allow to switch between the hamming
  # encoding plus the ascii and the ascii encoding only!

  if flag:

    print "Hamming and ascii encoding applied prior to the encryption"

    # define a dictionary to apply the hamming encoding of 4 bits

    dictionary_bin_to_code = {"0000":"0000000", "0001":"0001111",
    "0010":"0010110", "0011":"0011001", "0100":"0100101", "0101":"0101010",
    "0110":"0110011", "0111":"0111100", "1000":"1000011", "1001":"1001100",
    "1010":"1010101", "1011":"1011010", "1100":"1100110", "1101":"1101001",
    "1110":"1110000", "1111":"1111111"}

    # convert the data from
    # <class 'sage.monoids.string_monoid_element.StringMonoidElement'>
    # to <type 'str'>
    binary = str(ascii_to_bin(n))

    # loop that takes the binary input and slice it in every 4 bits

    # define an empty list to use in the loop
    hamming_encoded = str()

    # the slice operator [n:m] take the elements from n to m-1
    for i in range(0, len(binary)-3, 4):
      hamming_encoded += dictionary_bin_to_code[binary[i:i+4]]

  else:
    print "ONLY Ascii encoding applied prior to the encryption"
    hamming_encoded = str(ascii_to_bin(n))

  # convertion from a string to a list that is possible to multiply by q/2
  # lamdba allows to define "short" functions and map that iteratively
  # apply lambda to the list!

  hamming_encoded_int = map(lambda x: int(str(x)), list(hamming_encoded))

  return hamming_encoded_int

# ----------------------- # ----------------------- # -----------------------

# This SECOND script perform the encryption of a message.

# The operations are: c1 = a*e1 + e2, c2 = p*e1 +e3 +m

# This script behave like the normal encrypt but furthermore it takes any message
# and slices it in len(message)/n blocks and pads the last block with zeros.

# perform the encryption three polynomials e1, e2, e3 are needed. In the original
# construction all of them are sampled from a gaussian distribution.
# Sample e1 from a uniform (binary) distribution to make the encryption faster.
# Implementation: if with a flag: 0 classical construction,
# 1 (default) my construction.

# INPUT
# - n: size of the "lattice" or "degree +1" of the polynomials
# - q: modulus of the GF
# - std_dev: std deviation for the gaussian distribution
# - a: 1 part of the public key
# - p: 2 part of the public key
# - m: message -> a list of integers!
#----------
# OUTPUT
# - encrypted message

# -----modules for the sampling-----
from sage.crypto.lwe import UniformPolynomialSampler
from sage.stats.distributions.discrete_gaussian_polynomial
import DiscreteGaussianDistributionPolynomialSampler

# -----modules quotient ring operations-----
from sage.all import FiniteField
from sage.all import PolynomialRing

def encrypt_extended(n, q, std_dev, a, p, ascii_mess, flag_encryption = 1,
flag_hamming = 1):

  m = ascii_hamming_encode(ascii_mess, flag_hamming)

  num_calls = int(len(m)/n)
  num_pads = int(n - len(m)%n)

  # if the padding is needed the number of calls is increased:
  if len(m)%n != 0:
```

```
  num_calls += 1
  # adding zeros to m
  m += [0]*num_pads

# the input is a list of integers, the output will be 2 lists of integers
# the message needs to be represented as a polynomial

# define the polynomial ring that will be used to coerce the list of integers
R = PolynomialRing(FiniteField(q),'x')
x = R.gen()
Q = R.quotient(x**n +1, 'x')
x = Q.gen()

# encoder 1 -> q/2
enc = int((q-1)/2)

# define empty lists to return the list of ciphertexts
c1_list = []
c2_list = []

for i in range(num_calls):
  # slice the message:
  m_slice = m[i*(n):(i+1)*(n)]
  # coerce the list into polynomial in order to simply apply the multiplication
  m_poly = R(m_slice)*enc

  # generate the errors, if flag_encryption == 1 e1 \in uniform distribution {0,1}
  # else e1 is gaussian sampled; e2,e3 are gaussian sampled.

  if flag_encryption:

    print "The error e1 used in the encryption procedure have been generated
    according to a uniform distribution in {0,1}!!"
```

```
      e1 = UniformPolynomialSampler(Q, n, 0, 1)()
      e2 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()
      e3 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()
    else:

      print "The error e1 used in the encryption procedure have been
      generated according to a gaussian distribution!"

      e1 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()
      e2 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()
      e3 = DiscreteGaussianDistributionPolynomialSampler(Q, n, std_dev)()

    #ciphertext plus the gaussian noise
    c1_poly = a*e1 + e2
    c2_poly = p*e1 + e3 + m_poly

    c1_list += list(c1_poly)
    c2_list += list(c2_poly)

  # the flag_hamming is passed to the decryption algorithm to apply
  # the proper decoding prior to the ascii one

  return (c1_list, c2_list, flag_hamming)
```

A.3 decryption

```
# -----------------------
# SCRIPT BY ANTONIO VAIRA
# 16.12.2014
# -----------------------

# This FIRST script is responsible for the last stage of decoding:
# it takes a list of integers as input, slices it in sub-lists of
# 7 integers, apply the hamming decoding and finally reconstructs
# the original ascii encoded character!
# INPUT
# - list of integers with the binary encoded characters
#----------
# OUTPUT
# - ascii encoded string! the recovered original message within
# a certain probability

# Cut the padded zeroes, then slice the input list in 7bits lists,
# apply the code correcting algorithm, cutting the parity check bits,
# put together 2 4-tuples of bit and apply the bin_to_ascii encoding.

from sage.crypto.util import bin_to_ascii
from sage.coding.code_constructions import HammingCode
from sage.all import FiniteField

def ascii_hamming_decode(m_list, flag):

  if flag:
    # in case of an optimization the cutting value of the sub-lists
    # can be modified here and will affect all!
    message_length = len(m_list)
    cutting_value = 7
    extra_padding = message_length%(2*cutting_value)

    #cut the extra padding in the original message
    m_list = m_list[0:(message_length-extra_padding)]

    # two 7bits lists at time are needed
    num_calls = int(message_length/(2*cutting_value))

    # defines the code
    C = HammingCode(3, FiniteField(2))
    # defines an empty list
    _8bits_list = []

    for i in range(num_calls):
      #slices the message in 14bits lists
      m_list_slice = m_list[(i)*(2*cutting_value):(i+1)*(2*cutting_value)]

      #decodes two sublists and CUTS the parity check bits PCB and
      # append it to the _8bits_list outside the loop!
      _8bits_list += list(C.decode(m_list_slice[0:cutting_value]))[0:4] +
       list(C.decode(m_list_slice[cutting_value:2*cutting_value]))[0:4]

    return bin_to_ascii(_8bits_list)

  else:
    # I need to consider the extra padding also in this situation:
    # cut the encoded message padding here!
    message_length = len(m_list)
    extra_padding = message_length%8

    m_list = m_list[0:(message_length-extra_padding)]

    return bin_to_ascii(m_list)

# ----------------------- # ----------------------- # -----------------------

# This SECOND script define the basic decoder q/2 of the PKC

# INPUT
# - n: size of the "lattice" or "degree +1" of the polynomials
# - q: modulus of the GF
# - m_encoded: the polynomial corresponding to the decrypted message
# - window_parameter: the half-width of the decoding window around q/2
#  that corresponds to a decoded 1! due to the unbalanced Gaussians
```

```
#----------
# OUTPUT
# - coefficients: list of integers: message represented in bits

def decode(n, q, m_encoded, window_parameter):
  #it is possible to perform assignment only on the lists
  coefficients = map(lambda x: int(x), m_encoded)

  # WINDOW decoder - enhanced implementation

  for i in range(len(m_encoded)):
    if (coefficients[i] >= ((q-1)/2)-int(q*window_parameter))&(coefficients[i]
     <= ((q-1)/2)+int(q*window_parameter)):
      coefficients[i] = 1
    else:
      coefficients[i] = 0

#  # THRESHOLD decoder - original implementation
#  for i in range(len(m_encoded)):
#    if coefficients[i] >= ((q-1)/2):
#      coefficients[i] = 1
#    else:
#      coefficients[i] = 0

  return coefficients

# ----------------------- # ----------------------- # -----------------------

# This THIRD script perform the final decryption of a message:
# the message is a list of integers of arbitrary lenght.
# All the polynomial operations are handled by sage in the background:
# the script takes as input 2 list, slices them and embeds in the
# quotient polynomial ring.
# The decryption take as input the private key r2 and the
# ciphertext c1 and c2 and perform the simple operation: c1*r2 + c2.

# INPUT
# - n: size of the "lattice" or "degree +1" of the polynomials
# - q: modulus of the GF
# - std_dev: std deviation for the gaussian distribution
# - c1: 1 part of the ciphertext
# - c2: 2 part of the ciphertext
# - r2: the private key
# - window_parameter: the half-width of the decoding window around q/2
#  that corresponds to a decoded 1! due to the unbalanced Gaussians
#----------
# OUTPUT
# - decrypted message

# -----modules quotient ring operations-----
from sage.all import PolynomialRing
from sage.all import stats

def decrypt_extended(n, q, std_dev, c1_list, c2_list,
r2_poly, window_parameter, flag_hamming):

  # define the polynomial ring that will be used
  # to coerce the list of integers
  R = PolynomialRing(FiniteField(q),'x')
  x = R.gen()
  Q = R.quotient(x**n +1, 'x')
  x = Q.gen()

  #define the empty list to return - needed in the loop.
  m_encoded_list = []

  num_calls = int(len(c1_list)/n)
```

```
for i in range(num_calls):
  # slices the ciphertexts:
  c1_list_slice = c1_list[(i)*(n):(i+1)*(n)]
  c2_list_slice = c2_list[(i)*(n):(i+1)*(n)]

  # coerces the lists into the polynomial ring
  c1_poly = Q(c1_list_slice)
  c2_poly = Q(c2_list_slice)

  # performs the operations in the polynomial quotient ring
  m_encoded_poly = c1_poly*r2_poly + c2_poly

  # appends the list
  m_encoded_list += list(m_encoded_poly)


m_decode = decode(n, q, m_encoded_list, window_parameter)

m_ascii_decode = ascii_hamming_decode(m_decode, flag_hamming)

print "the ascii decoded message is: "
print m_ascii_decode

return (m_encoded_list, m_ascii_decode)
```

Bibliography

[1] Arora, S., Barak, B. *Computational Complexity: A Modern Approach*, Cambridge, 2009.

[2] Miklós Ajtai, *Representing Hard Lattices with O (n log n) Bits*, Chicago Journal OF Theoretical Computer Science 2, pp. 1-40, 2008.

[3] Babai, L., *On Lovász lattice reduction and the nearest lattice point problem.* Combinatorica, 6:1–13, 1986.

[4] Dave Bacon, *CSE 599d – Quantum Computing Grover's Algorithm*, Department of Computer Science & Engineering, University of Washington, Washington, 1996.

[5] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes* H. Krawczyk (Ed.): CRYPTO'98, LNCS 1462, pp. 26 - 46, 1998, Springer-Verlag Berlin Heidelberg, 1998.

[6] M. Bellare, P. Rogaway, *Optimal Asymmetric Encryption – How to encrypt with RSA*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.

[7] Bernstein Daniel J., *Cache-timing attacks on AES*, Department of Mathematics, Statistics, and Computer, The University of Illinois at Chicago, 2005.

[8] Daniel J. Bernstein, *Understanding brute force*, Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, 2005.

[9] Bernstein Daniel J., Tanja Lange, Christiane Peters, *Attacking and defending the McEliece cryptosystem*, Post-Quantum Cryptography, pp. 31-46, Springer Berlin Heidelberg, 2008.

[10] Bernstein Daniel J, Johannes Buchmann, and Erik Dahmen, *Post-quantum cryptography*, Springer Science & Business Media, 2009.

[11] Bernstein Daniel J, *List decoding for binary Goppa codes*, Coding and Cryptology, pp. 62-80, Springer Berlin Heidelberg, 2011.

[12] Campello Antonio, Grasiele C. Jorge, Sueli IR Costa, *Decoding q-ary lattices in the Lee metric*, arXiv preprint arXiv:1105.5557, 2011.

[13] Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede *Efficient Software Implementation of Ring-LWE Encryption*, Design, Automation and Test in Europe, 2015.

[14] Coglianese Michael, Bok-Min Goi, *MaTRU: A new NTRU-based cryptosystem*, Progress in Cryptology-INDOCRYPT, pp. 232-243, Springer Berlin Heidelberg, 2005.

[15] Damgard Ivan, Sunoo Park, *How Practical is Public-Key Encryption Based on LPN and Ring-LPN?*, Alekhnovich, FOCS 2003.

[16] Alessandra Di Pietro, Oliver Morsch, *Computer Quantistici*, Reviews of MONDO DIGITALE, N.48, pp.1-25 Milano, 2013.

[17] Léo Ducas, Alain Durmus, *Ring-LWE in polynomial rings*, Public Key Cryptography–PKC 2012. Springer Berlin Heidelberg, pp. 34-51, 2012.

[18] Dwarakanath Nagarjun C., Steven D. Galbraith, *Sampling from discrete Gaussians for lattice-based cryptography on a constrained device*, Applicable Algebra in Engineering, Communication and Computing 25.3: pp. 159-180, 2014.

[19] Frostig Roy, David Tobin, *Quantum Computing and Shor's Factoring Algorithm*, 2010.

code sign

[20] Gaborit, Philippe, Marc Girault, *Lightweight code-based identification and signature*, Information Theory, 2007. ISIT 2007, IEEE International Symposium on. IEEE, 2007.

[21] Steven D. Galbraith, *Space-efficient variants of cryptosystems based on learning with errors*, Mathematics Department, University of Auckland, New Zealand, 2013.

[22] Italo GHIDINI, *Codici per il controllo degli errori*, www.ding.unisannio.it/it/persone/docenti-a.../italo-ghidini, A.A. 2013-2014.

[23] Oded Goldreich, Shafi GoldWasser, Shai Halevi *Public-Key Cryptosystems from Lattice Reduction Problems*, CRYPTO '97 Proceedings of the 17th Annual International

Cryptology Conference on Advances in Cryptology pp.112 – 131, Springer-Verlag London, UK, 1997.

[24] Robert M. Gray, *Toeplitz and Circulant Matrices: A Review*, Foundations and Trends® in Communications and Information Theory 2.3, pp.155-239, 2005.

[25] L. K. Grover, *A fast quantum mechanical algorithm for database search.*, 28th Annual ACM Symposium on the Theory of Computing, pp. 212-219, 1996.

[26] Robert M. Grow, Thomas Prevost, et al, *IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices*, The Institute of Electrical and Electronics Engineers, Inc., in the United States of America, 2009.

[27] David Harvey, *Faster arithmetic for number-theoretic transforms*, Journal of Symbolic Computation 60, pp. 113-119, 2014.

[28] Howard M. Heys, *A tutorial on linear and differential cryptanalysis*, Cryptologia 26.3 (2002): pp. 189-221, 2002.

[29] Hoffstein Jeff, et al, *NTRU: A public key cryptosystem*, URL: http://grouper.ieee.org/groups/1363/lattPK/submissions/ntru. Pdf, 1999.

[30] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H Silverman, Wiliam Whyte, *Hybrid Lattice Reduction and Meet in the Middle Resistant Parameter selection for NTRUEncrypt*, Advances in Cryptology-CRYPTO, pp. 150-169. Springer Berlin Heidelberg, 2007.

[31] Howgrave-Graham, Nick. *A hybrid lattice-reduction and meet-in-the-middle attack against NTRU*, Advances in Cryptology-CRYPTO 2007, pp. 150-169, Springer Berlin Heidelberg, 2007.

[32] Ellen Jochemsz *Goppa Codes & the McEliece Cryptosystem*, Divisie Wiskunde en Informatica, Universiteit Amsterdam, 2002.

[33] Phillip Kaye, Raymond Laflamme, Michele Mosca *An Introduction to Quantum Computing* OXFORD University Press, 2007.

[34] A. K. Lenstra, H. W. Lenstra, and L. Lovász, Factoring polynomials with rational coefficients, Mathematische Annalen, 261(4):515–534, 1982.

[35] Karu Priit, Jonne Loikkanen, *Practical comparison of fast public-key cryptosystems*, Telecommunications Software and Multimedia Lab. at Helsinki Univ. of Technology, Seminar on Network Security, Citeseer, 2001.

[36] Vadim Lyubashevsky, Daniele Micciancio,Chris Peikert, and Alon Rosen. *SWIFFT: A modest proposal for FFT hashing*, Fast Software Encryption, Springer Berlin Heidelberg, 2008.

[37] Richard Lindner, Chris Peikert *Better key sizes (and attacks) for LWE-based encryption*, Topics in Cryptology–CT-RSA 2011. Springer Berlin Heidelberg, pp. 319-339, 2011.

[38] Lyubashevsky Vadim, Chris Peikert, Oded Regev, *On ideal lattices and learning with errors over rings*, Journal of the ACM (JACM) 60, p. 43, 2012

[39] Lyubashevsky Vadim, Chris Peikert, Oded Regev, *A Toolkit for Ring-LWE Cryptography*, EUROCRYPT, Vol. 7881, pp. 35-54, 2013.

[40] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. *On Ideal Lattices and Learning with Errors Over Rings*, Journal of the ACM 60.6, 2013.

[41] McEliece Robert J., *A public-key cryptosystem based on algebraic coding theory*, DSN progress report 42.44, pp. 114-116, 1978.

[42] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone *HANDBOOK of APPLIED CRYPTOGRAPHY* MIT press, 1996

[43] Micciancio Daniele, Bogdan Warinschi, *A Linear Space Algorithm for Computing the Hermite Normal Form*, Proceedings of the 2001 international symposium on Symbolic and algebraic computation, ACM, 2001.

[44] Micciancio Daniele, *Improving lattice based cryptosystems using the Hermite normal form*, Cryptography and Lattices, pp. 126-145, Springer Berlin Heidelberg, 2001.

[45] Micciancio Daniele, *Generalized compact knapsacks, cyclic lattices, and efficient one-way functions*, Computational Complexity 16.4 (2007): 365-411, 2007.

[46] Daniele Micciancio, Chiris Peikert, *Hardness of SIS and LWE with Small Parameters*, CRYPTO, Volume 8042, p.21-39, LNCS, 2013.

[47] Oliver Morsch, *Crittografia Quantistica*, CNR-INFM, Dipartimento di Fisica, Pisa, 2014.

[48] Michael Naehring *Homomorphic Encryption from Ring Learning with Errors*, Technische Universiteit Eindhoven, michael@cryptojedi.org, MSR Cambridge, 2012.

[49] Nguyen Phong Q., Oded Regev, *Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures*, Advances in Cryptology-EUROCRYPT 2006, pp. 271-288 Springer Berlin Heidelberg, 2006.

[50] Chris Peikert, Alon Rosen, *Efficient Collision – Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices*, Theory of Cryptography, p. 145-166 Springer, Berlin Heidelberg, 2006.

[51] Chris Peikert, *Lattice-Based Cryptography: Short Integer Solution (SIS) and Learning With Errors (LWE)*, Georgia Institute of Technology, Atlanta, USA, 2013.

[52] Geovando Carlos CF Pereira, Paulo SLM Barreto, *Introduction to Multivariate Public Key Cryptography*, Escola Politécnica, University of Sao Paulo, 2013.

[53] Abhilash Ponnath, *Difficulties in the Implementation of Quantum Computers*, Cornell University Library, 2006 `arxiv.org/pdf/cs/0602096`.

[54] Pöppelmann Thomas, Tim Güneysu, *Towards practical lattice-based public-key encryption on reconfigurable hardware*, Selected Areas in Cryptography–SAC 2013, pp. 68-85, Springer Berlin Heidelberg, 2014.

[55] Regev, O., *On lattices, learning with errors, random linear codes, and cryptography.* In Proc. 37th ACM Symp. on Theory of Computing (STOC), pages 84–93, 2005.

[56] Oded Regev, *New lattice-based cryptographic constructions*, Journal of the ACM (JACM) 51.6 (2004): pp. 899-942, 2004.

[57] Oded Regev, *Quantum computation and lattice problems*, SIAM Journal on Computing 33.3 (2004): pp. 738-760, 2004.

[58] Steven Rich, Barton Gellman, *"NSA seeks to build quantum computer that could crack most types of encryption".*, Washington Post., January 2, 2014.

[59] R.L. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Martin Gardner's Scientific American column, 1977.

[60] Roy, Sujoy Sinha, et al, *Compact Ring-LWE Cryptoprocessor*, 2014.

[61] Michael Rose, *Lattice-based cryptography: a practical implementation*, University of Wollongong, 2011.

[62] W. A. Stein et al., *Sage Mathematics Software (Version 4.6.1)*, The Sage Development Team, 2014, `http://www.sagemath.org`.

[63] Tourloupis, Vasilios Evangelos, *Hermite normal forms and its cryptographic applications*, University of Wollongong, 2013.

[64] WANG Hou-Zhen, ZHANG Huan-Guo, *Hash-based Multivariate Public Key Cryptosystems*, The Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Computer, Wuhan University, Wuhan 430072, wanghouzhen@126.com, China, 2011.