Universiteit Leiden
The Netherlands

Opleiding Informatica en Economie

A survey of classical public-key cryptography and post-quantum lattice-based cryptography

Vinay Oedayrajsingh Varma

Supervisors:
dr. A.H. Deutz
dr. W.A. Kosters

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

25/08/2017

Abstract

This thesis is a survey of some cryptographic systems used in public-key cryptography. An introduction to systems in classical public-key cryptography is given as well as reasons of why the security in these are potentially breached. Notable examples we discuss are Diffie-Hellman and RSA, both of which have been proven not to be resistant to quantum attacks. A brief explanation is given of why these two system are not resistant to quantum attacks. To find promising cryptographic systems which are quantum immune, we introduce lattice-based cryptography with notable examples such as the GGH cryptosystem. Along with the introduction of these systems, we will also describe the computational hardness of the cryptographic primitives. Furthermore, explanations of general concepts such as provable security and basis reduction in lattice-based cryptography will be given.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cryptography is one of the fundamentals of internet security. In the present-day era there are many sorts of applications where formal cryptographic structural procedures (algorithms) are present. For instance, bank ATMs use cryptographic algorithms to make sure a customer's bank transaction is secure. Also, the vote of a person is encrypted in electronic voting to make sure the data (e.g., vote) cannot be changed by a third party. The security of cryptographic algorithms is based on hard-to-solve mathematical problems and is tied to the computational power in present-day computers. The cryptographic algorithms are designed in such a way that any attack by a classical computer on any cryptographic algorithm is deemed infeasible.

Quantum computers have much higher computational power and break many present-day cryptographic algorithms. Though currently still in its infancy, companies and research centres around the world acknowledge the potential of quantum computing and invest much money progressively advancing towards a possible prototype. An attack by a quantum computer could break many present-day cryptographic algorithms, essentially meaning that they could break many channels which were thought be to secure. There is, however, a selection of algorithms which, with possibly little tweaks, are not broken by quantum computers.

Post-quantum cryptography describes these algorithms, it is essentially the search for algorithms which are thought to sustain attacks from a quantum computer. With the abundance of proven classical cryptographic algorithms that can maintain security integrity on a classical level, one could ask if this level of security is maintained on a quantum level.

In this thesis, parts of a subject within post-quantum cryptography is explained, namely lattice-based cryptography. A survey of lattice-based cryptography and its primitives is given as well as an overview of which components of this post-quantum cryptography subject are used in quantum algorithms.

1.1 Research survey

This thesis concludes a research survey done on the fields of the classical based public-key cryptography and lattice-based cryptography. It denotes the parts of each of the algorithms as well as a decomposition of primitives used to construct them along with actual examples.

1.2 Thesis setup and overview

The thesis will contain much information regarding quantum computing and post-quantum cryptography. Because of these intangible subjects many experiments will be done through thought experiments. Originally predominantly done in physics, thought experiments describe experiments done not in reality but in a hypothetically setup scenario.

Furthermore, general explanations, contextual information and examples regarding cryptography and quantum computing will be given.

The following chapters will give insight into the classical definitions of cryptography, contextual information and explanations of two commonly used classical algorithms in cryptography. Furthermore, parts of lattice-based cryptography with some algorithms will be explained, illustrated with some lattice-based cryptography examples.

1.3 Project

This research project is a bachelor's thesis and concludes the bachelor education program in Computer Science & Economics at the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University. The thesis is supervised by dr. A.H. Deutz and dr. W.A. Kosters.

Chapter 2

Definitions

This chapter contains contextual information about classical computing, quantum computing and cryptography in general. Furthermore, two classical examples of cryptographic algorithms (RSA and Diffie-Hellman) are given as well as a brief explanation of why these algorithms are broken by quantum computers.

2.1 Classical computing and quantum computing

Classical computers store information through bits. They are defined to be either 0 or 1. The quantum equivalent of these bits are called qubits. Qubits exhibit the quantum behaviour of entanglement and superposition, in such a way that they are defined to be both 0 and 1 at the same time when not measured. When measuring a qubit it chooses a "side" which is either 0 or 1. This quantum ability has a big advantage. As postulated by Richard Feynman in 1982, any processing done to a qubit contains the output of a qubit in superposition [4].

An operation on a classical bit gives the output of a single bit, it represents one of two disjoint situations [5]. An operation on a qubit gives the output of a single qubit. The single qubit has more output with the same amount of operation. Extending this to two bits, a classical computer will have to do four operations to get the result 00, 01, 10, 11. Scaling up even more, it is clear that the increase in operations grows exponentially.

Application-wise, this means that certain mathematical problems which were originally computed (or limited) through classical computers could now be easier to solve. Since the principles of cryptography are based on hard-to-solve mathematical problems, new developments in quantum computing could break these hard-to-solve mathematical problems, thus representing a threat to encryption. Naturally, only the cryptographic algorithms that use these hard-to-solve mathematical properties at their basis are affected. However, studying the concept of cryptography and the set of rules of making cryptographic primitives, it is clear that it is not so easy to set up a new quantum cryptographic primitive or even design new cryptographic schemes that can exhibit the same principles as the regular classical algorithms [1].

2.2 Cryptography

In cryptography the art of secure communication through public channels among adversaries is studied. The most notable parts of cryptography are encryption and decryption. Encryption is the mapping of a certain message to a ciphertext message using a particular key. Decryption is the reverse procedure. That is, reversing the mapping from ciphertext back to the original text. These encryption/decryption methods are incorporated in encryption/decryption algorithms. These algorithms are part of the cryptographic algorithms. There are three main types of cryptographic algorithms:

- Symmetric (Secret-key) cryptography
- Asymmetric (Public-key) cryptography
- Cryptogaphic hash functions

Each of these general cryptography types have their own domain. For example, public-key algorithms are mostly used for authentication whereas secret key cryptography is more prominently used for privacy and confidential information.

At the basis of these cryptographic algorithms are the cryptographic primitives. These primitives are designed in such a way that an adversary using the system cannot breach any of the set security goals in any way. In these cases, it is assumed that the adversary has a reasonably powerful classical computer limited by randomized polynomial time [1].

One could ask however, what would happen if the adversary, instead of a classical computer, had access to a reasonably powerful quantum computer. Thought experiment-wise, one could already assume that there would be many algorithms broken by this reasonably powerful quantum computer. The most used cryptogaphic algorithms like RSA [6], Diffie-Hellman [7] and AES (also its predesessors DES and Triple-DES), all of which have classically computational hard-to-solve properties in their primitives, would be broken. In Table 2.1 an overview is given of cryptographic algorithms along with their types and whether or not they are broken by a quantum algorithm [1,8].

Table 2.1: Overview of some popular cryptographic algorithms which are (not yet) broken by a quantum computer [1]

Cryptographic system	Broken by quantum algrithms?
RSA (public-key)	yes
Diffie-Hellman (public-key)	yes
Buchmann-Williams (public-key)	yes
Algebraically Homomorphic (hash)	yes
McEliece (public-key)	not yet
NTRU (public-key)	not yet
Lattice-Based (public-key)	not yet

2.2.1 Computational hardness

As previously mentioned, cryptographic encryption methods are based on hard-to-solve mathematical problems. These hard-to-solve problems are defined under the computational hardness assumption. Many cryptographic algorithms, in particular the algorithms associated with asymmetric cryptography, are based on mathematical problems for which currently no efficient solution is available. These problems are based on one-way functions. Functions like these can easily construct a new large piece of several smaller pieces, but decomposing this large piece into the original smaller pieces is considered to be very hard. Note that there is a big gap between finding a computationally hard problem and using this problem to construct a useful cryptographic primitive.

The definition of easy and hard here should be viewed from a complexity point of view. Namely, complexity hard-to-solve problems are considered hard because currently there is no efficient solution available. The current (non-efficient) solutions are then asymptotically limited by some function which describe a fit approximation of that function. There are some notable examples in this computational hardness domain which are much used in cryptography algorithms:

- Integer factorization problem
- Discrete logarithm problem
- Elliptic curve problem

These problems are part of a mathematical research field called the hidden subgroup problem. Also, the shortest vector problem, which will be discussed later in the lattice-based cryptography chapter, is an example which could be reduced to the hidden subgroup problem.

The previously mentioned cryptographic algorithms RSA and Diffie-Hellman both rely on problems defined in the computational hardness domain. More specifically, RSA relies on the hardness of factoring integers and Diffie-Hellman relies on the hardness of discrete logarithms. The two mentioned hardness assumptions are part of two distinct families defined in public-key cryptography, namely the logarithm family and factoring family. Elliptic curve cryptography is also part of the logarithm family as its property is just defined on finite fields instead of modular mathematics.

2.2.2 Worst-case hardness vs average-case hardness

When working on cryptography, we describe worst and average type cases. Worst-case hardness problems use the maximum amount of computer resources when running algorithms. Average-case hardness gives a more accurate view of a cryptographic algorithm's performance as it does not consider input which would result in intractable cases. Hence, average-case hardness problems use average amount of computer resources.

In the field of complexity, worst-case hardness is generally the most common measure of hardness [9]. However, when moving to cryptography, average-case hardness is needed. When looking at the Diffie-Hellman cryptosystem, which is based on the discrete logarithm problem, the algorithm is only so strong because the discrete logarithm problem is a hard problem to solve for the average-case. It would not be secure at all if it would be hard to solve in the worst-case, meaning general cases of discrete logarithm problems would be easy to solve.

This does not mean however, that worst-case problems are not important. This will also later be seen in lattice-based cryptography in chapter 3. Worst-case complexity is needed to *understand* the formulated problem and average-case complexity is needed to *solve* the formulated problem. Since any instance in the worst-case of the problem is solvable given enough resources, also the hardest instance will be. For random instances the solvability is random. Figure 2.1 gives an overview of solvable instances in worst-case hardness and solvable instances in average-case hardness. As stated, working with cryptography requires average-case hardness as you want the random instance to be hard to solve.

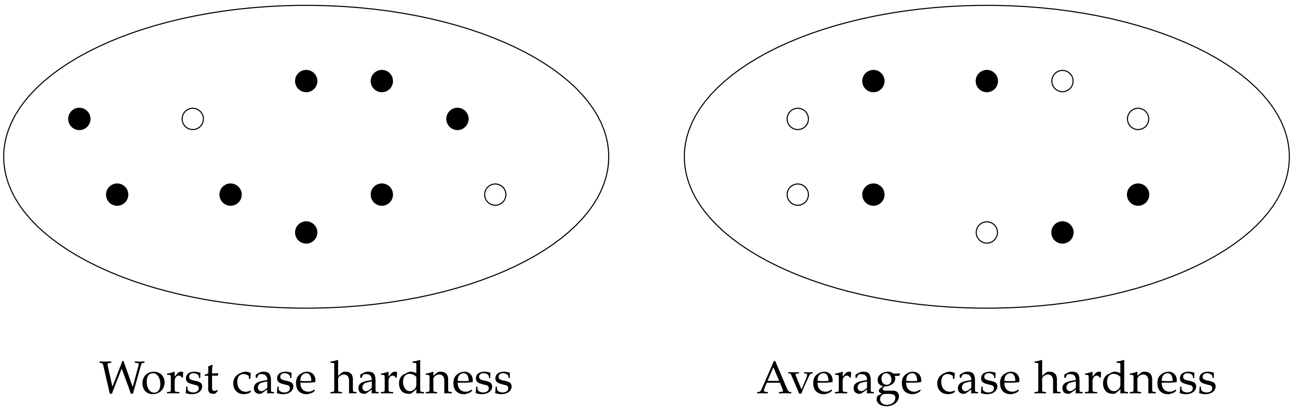

Figure 2.1: Overview of instances in worst-case hardness and average-case hardness scenarios. At least one instance (open circle) is hard (left) and average-case instances (open circle) are hard (right).

2.3 Diffie-Hellman key exchange

The Diffie-Hellman (D-H) method is one of many asymmetric cryptography methods [7]. It got particularly popular because it was also one of the first widely used methods. The D-H key exchange method lets users exchange a secret key by means of communicating through a public channel. As stated before, D-H relies on the computational difficulty of discrete logarithms which is part of the hidden subgroup problem. Multiplying two large input size integers can be done very quickly, whereas factoring integers with large input size takes very long. The essence of the discrete logarithm problem is that computing the inverse of modular arithmetic is considered not feasible to do even when having access to all of today's computational power.

D-H uses a prime modulus p and a generator g which is a primitive root of the prime modulus p. This has an important property, namely the fact that when raised to exponent x, the solution is uniformly congruent to any numbers between 0 and p. For example, taking a prime modulus $p = 13$ and primitive root generator $g = 6$. Calculating $g^x \bmod p$ is uniformly congruent to all numbers between 0 and 13 with equal probability.

The D-H key-exchange in its simplest form describes the process of agreeing upon a secret message through a public communication channel making use of the above mentioned modular arithmetic property. Consider a situation where person A called Alice wants to send a message to person B called Bob through a public channel. In this public channel there is always an eavesdropper Eve who is interested in this message. Figure 2.2 provides a schematic overview of such a system. We will be using this basic system more often in the sequel.

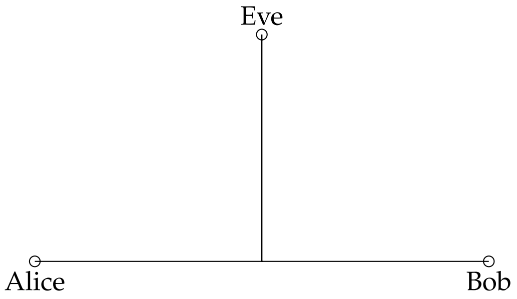

Figure 2.2: Basic three person communication system.

2.3.1 Example

Consider the basic communication system in Figure 2.2 introduced above. As explained above, Alice and Bob want to communicate a shared secret over a public channel. They agree publicly on a prime modulus $p = 13$ and a generator $g = 6$. This results in a publicly known number 6 mod 13. Both Alice and Bob then choose a private random number. In this case, Alice chooses 9 and Bob chooses 10.

Alice computes 6^9 mod 13, using the publicly known number 6 mod 13 and her private number 9 as her exponent; 6^9 mod $13 = 5$.

Bob computes 6^{10} mod 13, also using the publicly known number 6 mod 13 and his private number 10 as his exponent; 6^{10} mod $13 = 4$.

Both Alice and Bob then send their solutions through the public channel to each other. Figure 2.3 denotes the postulated situation along with a description of known information. Also, Table 2.2 denotes which information is public and private. Publicly known are 6 mod 13, 4 and 5.

Table 2.2: Publicly and privately known information in D-H key exchange system.

	Alice	Bob	Eve
public	6 mod 13 using 6^x mod $13 = 4$ 6^x mod $13 = 5$		
Private	9	10	-

Now that both Alice and Bob have each other's number, they can determine the shared secret. Alice's computation involves taking Bob's sent number as the base, and raising it to her private number 9 modulo 13. As such, she determines the shared secret to be 4^9 mod $13 = 12$. Independently, Bob determines the shared

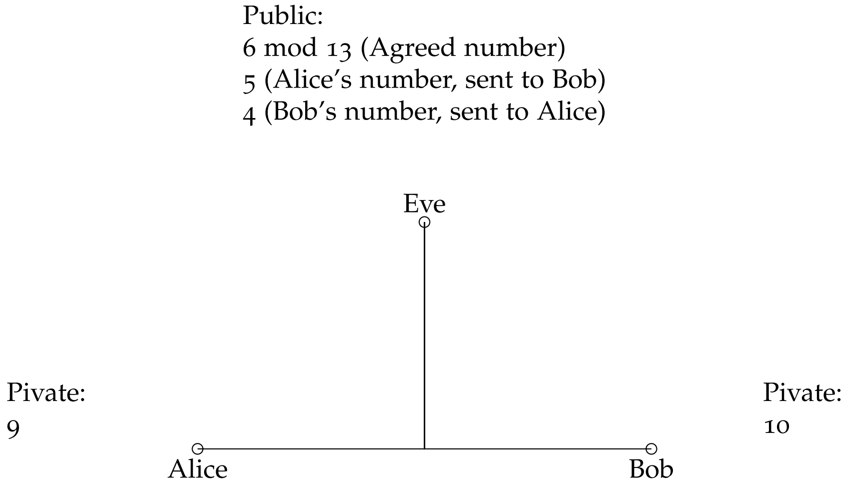
Public:
6 mod 13 (Agreed number)
5 (Alice's number, sent to Bob)
4 (Bob's number, sent to Alice)
Eve
Pivate:
9
Pivate:
10
Alice
Bob

Figure 2.3: Basic three person communication system with public and private information in D-H example.

secret by taking Alice's sent number as the base, and raising it to his private number 10 modulo 13. As such, Bob determines the shared secret to be $5^{10} \bmod 13 = 12$. Note that both Alice and Bob have found the same shared secret 12 by communicating different numbers through a public channel.

Though not directly visible, Alice and Bob essentially both did the same calculation. Note that the sent number 4 from Bob was derived from the calculation $6^{10} \bmod 13 = 4$. Alice has actually performed the calculation $\left(6^{10}\right)^9 \bmod 13 = 12$. Similarly, the sent number number 5 by Alice was derived from the calculation $6^9 \bmod 13 = 5$. Hence, Bob performed the computation $\left(6^9\right)^{10} \bmod 13 = 12$. Note that $\left(6^{10}\right)^9 \bmod 13 = 12 = \left(6^9\right)^{10} \bmod 13$. This is because of the trivial property $\left(A^b\right)^c = A^{bc}$ and $\left(A^c\right)^b = A^{cb}$.

Suppose the adversary Eve wants to know the shared secret. The publicly known information consists in this case of the publicly shared number 6 mod 13 and the two sent numbers 4 and 5. For Eve to discover the shared secret 12, Eve has to solve the discrete logarithm problem, calculating $6^x \bmod 13 = 5$ or $6^x \bmod 13 = 4$. Now, because these are small numbers the calculation can be performed relatively easily by trial and error. However, scaling up the used numbers, it becomes infeasible to do the computations by trial and error, resulting in a secret only known to both Alice and Bob and not to Eve.

2.4 RSA cryptosystem

In 1977 Rivest, Shamir and Adleman introduced the asymmetric RSA algorithm [6]. It is currently the most widely used algorithm in the world primarily used for encrypting data traffic [10]. RSA relies on the computational hardness of factoring integers. More specifically, factoring composite numbers. A one-way function considers that it is easy to construct a number from two prime numbers but hard to factor the generated number into the two originally used prime numbers. See the graph in Figure 2.4 for an overview of multiplying integers and factoring them with time. Note how for even large input numbers the multiplication time is essentially always a few seconds whereas factoring them would take years.

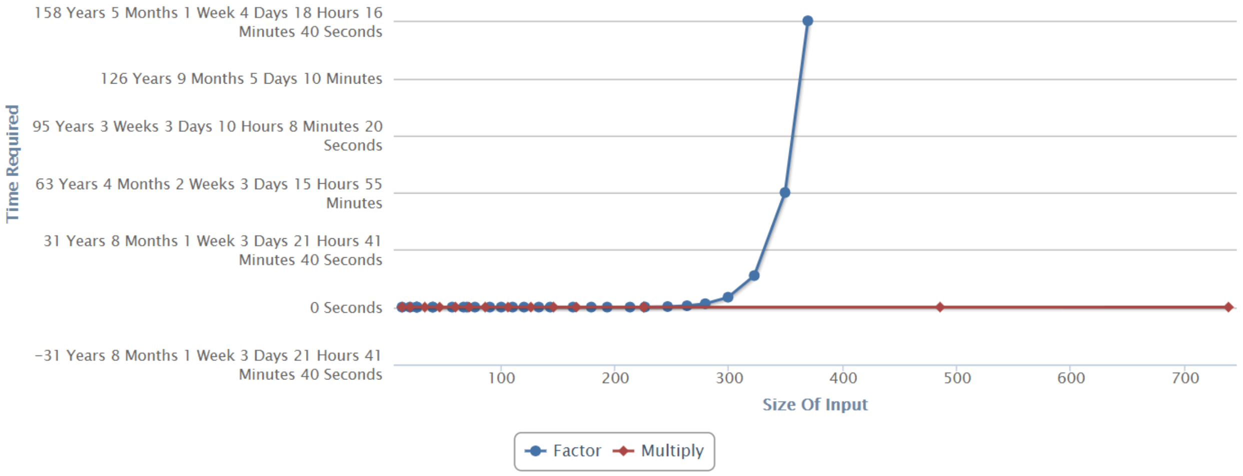

Figure 2.4: overview of multiplication time and factorizing time with larger size input — generated using [2].

In the three person system presented in Figure 2.2, we consider that Bob wants to send messages to Alice without eavesdropper Eve knowing what the message contains. We distinguish four steps in the RSA algorithm:

1. Generate keys, public and private
2. Distribute public key
3. Encryption using public key
4. Decryption using private key

Generating the public and private key is done by anyone who wants to receive a message. In this case we consider Alice in our three person system. The way that the key is produced is only known by Alice, as she is the one determining the key. It involves a multiplication of two prime numbers p and q to determine a composite number n. Alice also determines the Euler's totient Φ of composite number n. Since the Euler's totient of any prime number p is $p-1$ and Euler's totient function is multiplicative, the Euler's totient of composite number n can be simply determined [11]. We know that $n = p \times q$, using the multiplicative property we can determine $\Phi(n)$ to be $\Phi(n) = \Phi(p) \times \Phi(q)$. Since p and q are prime numbers, Euler's totient says that $\Phi(p) = p - 1$ and $\Phi(q) = q - 1$ resulting in

$$\Phi(n) = (p-1) \times (q-1) \tag{2.1}$$

Furthermore, Alice chooses e such that

$$\gcd(e, \Phi(n)) = 1.$$

.

Subsequently, she computes d such that

$$e \times d = 1 \bmod \Phi(n). \tag{2.2}$$

d exists by virtue of Bézout's identity and can be found using the extended Euclid's algorithm. Note that equation 2.2 can also be written as

$$e \times d = k \times \Phi(n) + 1 \quad \text{with } k \in \mathbb{Z} \quad \text{and} \quad e \text{ not sharing a factor with } \Phi(n) \tag{2.3}$$

here, d is part of the trapdoor to undo the effect of e.

The public can encrypt any message m using n and e to ciphertext c

$$c = m^e \bmod n \tag{2.4}$$

Euler's totient can be connected to modular exponentiation using Euler's theorem.

$$m^{\Phi(n)} = 1 \bmod n \qquad \text{with } m, n \text{ co-prime} \tag{2.5}$$

Since $1^k = 1$ with $k \in \mathbb{Z}$ and $\left(A^b\right)^c = A^{bc}$ we can also write

$$m^{k \times \Phi(n)} \equiv 1 \bmod n$$

Also, since $1 \times m = m$, multiplying both sides of the equation with m, we can write

$$m \times m^{k \times \Phi(n)} = m \bmod n$$

which can be simplified as

$$m^{k \times \Phi(n) + 1} = m \bmod n \tag{2.6}$$

Alice needs some information to reverse the encryption done by others. She keeps the private key d which we determined earlier to undo the effect of e.

$$(m^e)^d \bmod n = m$$

This simplifies to

$$m^{ed} \bmod n = m$$

She hides the numbers d, p, q and k, which are used for decryption, in her trapdoor. Decrypting any ciphertext c she computes

$$m = c^d \bmod n \tag{2.7}$$

note that d was computed using p and q. See Table 2.3 for an overview of publicly and privately known information.

Table 2.3: Overview of publicly and privately known information in RSA algorithm example

Publicly known	Privately known (trapdoor)
n, e	d, p, q, k

2.4.1 Example

Consider our basic three-person communication system in Figure 2.2 which we introduced in Section 2.3. Suppose Bob wants to send Alice a certain message m which was converted by Bob into a number using some kind of arbitrary translation scheme. Here, we consider Bob wants to send a message $m = 27$. For Alice to receive a ciphertext c from her senders, which in this case is only one sender Bob, she randomly selects two prime numbers $p = 13$ and $q = 17$. She determines n which is $n = p * q = 13 * 17 = 221$. She also determines $\Phi(n)$ using equation 2.1; $\Phi(n) = 12 * 16 = 192$. Alice determines e which, as seen in equation 2.3, must not share a factor with $\Phi(n) = 192$. She determines $e = 5$. Now d can be computed using equation 2.3;

$$d = \frac{k * \Phi(n) + 1}{e} = \frac{2 * 192 + 1}{5} = 77$$

Now Alice can hide $d = 77$, $p = 13$, $q = 17$ and $k = 2$ in her trapdoor and make $n = 211$ and $e = 5$ publicly available for encryption.

Bob encrypts his message $m = 27$ using equation 2.4. He determines his ciphertext c to be

$$c = 27^5 \bmod 211 = 40$$

and sends this publicly to Alice over the communication channel. Figure 2.5 gives an overview of the current situation.

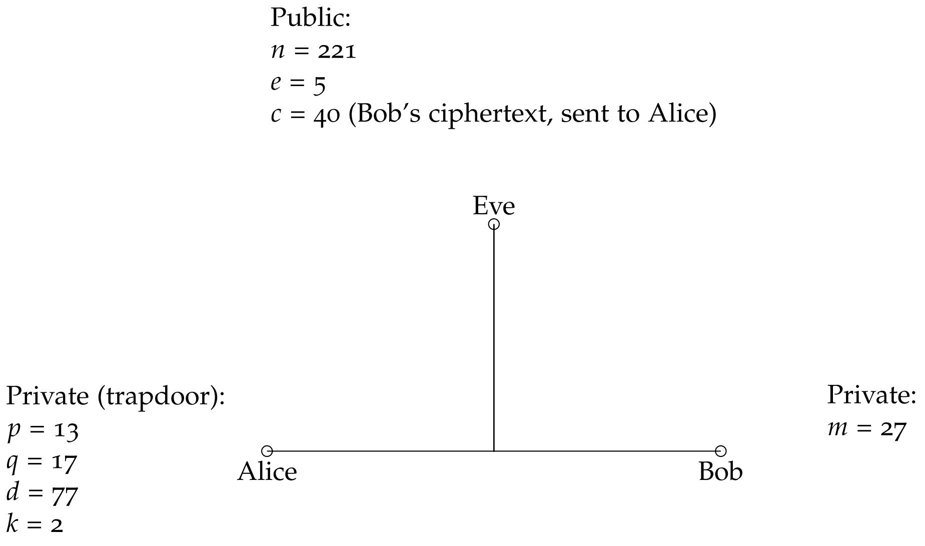

Figure 2.5: Basic three person communication system with public and private information in RSA example.

Alice receives Bob's ciphertext $c = 40$ and decrypts the text using the key information from her trapdoor. She decrypts the ciphertext c using equation 2.7 which results in

$$m = 40^{77} \bmod 221 = 27$$

which is the original message Bob sent.

Note that the adversary Eve has to do the same computation without knowing d. She tries $m = 40^d \bmod 221$. For Eve to determine d, she will have to factor the composite number n. As we have seen in Figure 2.4, factorizing the number n could take many years when choosing p and q with sufficiëntly large input size. This property makes sure the message Bob sends Alice is only known by Alice and not practically knowable by Eve.

2.4.2 Why it works

To prove why RSA works we will define two theorems: Fermat's little Theorem and the unique factorization theorem. Fermat's Little Theorem and the unique factorization theorem are defined below.

Fermat's Little Theorem (FLT)
For any given $x \in \mathbb{Z}$ and p being prime, it holds that $x^p = x \bmod p$.
With the additional condition that x is not a multiple of p we can divide both sides by x and get $x^{p-1} = 1 \bmod p$.

Unique Factorization Theorem (UFT)
Any integer ≥ 1 can be written as product of prime powers.

Given he following statements:

$N = p \times q$ with p and q distinct prime
$\Phi(n) = (p-1) \times (q-1)$
e is chosen such that $1 > e > n$ and e and $\Phi(n)$ co-prime, that is $\gcd(e, \Phi(n)) = 1$
d is chosen such that $e \times d = 1 \bmod \Phi(n)$
m is chosen such that $0 \leqslant m < n$
$c = m^e \bmod n$

we claim that

$$m = c^d \bmod n$$

is true.

Part A
We start off by proving that $m^{e \times d} \bmod p = m \bmod p$.

$m^{e \times d} \bmod p =$
because $e \times d = 1 \bmod \Phi(n)$, it holds that $e \times d = 1 + k \times \Phi(n)$ for every $k \in \mathbb{Z}$, substituting this equation gives us

$m^{k \times \Phi(n)+1} \bmod p$

we can rewrite this to

$m^{k \times \Phi(n)} \times m \bmod p$

since $\Phi(n) = (p-1) \times (q-1)$, we can write

$m^{k(p-1)(q-1)} \times m \bmod p$

rearranging gives us

$(m^{(p-1)})^{q-1} \times m \bmod p$

distributing the modulus gives us

$(m^{(p-1)} \bmod p)^{q-1} \times m \bmod p$

Now we consider two situations:

1. m and p are co-prime. That is, $\text{GCD}(m, p) = 1$.
2. m and q are not co-prime, meaning they share a common divisor.

situation 1

In the case that m and p are co-prime:

$m^{e \times d} \bmod p =$

By Fermat's little Theorem which states that $m^{p-1} = 1 \bmod p$, we can see that

$(m^{(p-1)} \bmod p)^{q-1} \times m \bmod p = m \times 1^{k^{q-1}} \bmod p$

Since 1 to the power of anything is always 1, we get

$m \bmod p$, thus concluding our part that $m^{e \times d} \bmod p = m \bmod p$

situation 2

In the case that m and p are not co-prime. That is, they share a certain common factor t. Then it must hold that

$m = t \times p$

but since we know that p is prime, dividing m by p (with $m > p$) will always result in a modulus of 0. Thus, we get

$m \bmod p = 0$

Naturally, this statement also holds when we raise m to any power.

$m^{e \times d} \bmod p = 0$

Thus we get again.

$m^{e \times d} \bmod p = m \bmod p$

We now have proved that $m^{e \times d} \bmod p = m \bmod p$ is true for p. Analogue to the whole process described in part A we can also prove this for q. This results is $m^{e \times d} \bmod q = m \bmod q$

Now we proceed to the next step, proving that $m = c^d \bmod n$ is true.

Part B

$m = c^d \bmod n$ is true.

$m^{e\times d} \bmod p = m \bmod p$ as described in part A

$m^{e\times d} \bmod q = m \bmod q$ as described being analogous to part A us

moving m to the left side results in the right side being 0 mod p, meaning there is a certain constant, say t, that is a multiple of p. Symmetrical for q.

$m^{e\times d} - m = t_1 p$

$m^{e\times d} - m = t_2 q$

Combining the above two statements and the UFT we obtain a new constant factor T such that

$m^{e\times d} - m = T \times pq$

From the property that $n = p \times q$, we substitute pq with n resulting in

$m^{e\times d} - m = T \times n$

Introducing modulus operation again, we get

$m^{e\times d} \bmod n = m \bmod n$

Given that m is chosen such that $0 \leqslant m < n$, $m \bmod n$ will always evaluate to m.

$m^{e\times d} \bmod n = m$

Rewriting the modulus gives us

$m = (m^e \bmod n)^d \bmod n$

Substituting $c = m^e \bmod n$ in the equation gives us

$m = c^d \bmod n$

This concludes our proof that $m = c^d \bmod n$ is true. ■

2.5 Shor's algorithm

In 1994 Peter Shor introduced a quantum algorithm that could break a large portion of public-key cryptography [12]. He described an algorithm that could efficiëntly solve both the integer factorization problem and the discrete logarithm problem. The strength of Shor's algorithm is that it runs in *polynomial time* on a quantum computer. More specifically, classical factoring algorithms run in $\mathcal{O}(\log N^k)$ whereas the quantum counterpart runs in $\mathcal{O}(\log N)$ [12]. As a result, any cryptographic algorithm that uses either integer factorization or discrete logarithms as its primitive is broken by Shor's algorithm when sufficiently large quantum computers are readily available. Notably, this includes both of the above explained algorithms Diffie-Hellman and RSA.

Shor's algorithm uses three main components in its algorithm: Modular arithmetic, Quantum parallelism and Quantum Fourier transformation. As we have seen in the example of RSA, knowing the prime composition of the large composite number N would essentially give Eve the ability to break Bob's ciphertext c. Shor's algorithm gives an adversary the possibility to find this composition of the composite number N. The formal problem is:

Given a composite number N, find the integer l, which is strictly between 1 and N, such that l divides N.

Essentially, we input a composite number N and output a prime number l which is a non-trivial factor of the inputted composite number N. In the algorithm we describe five steps to find l, four of which can be done using a classical computer and one, namely step 2, which is done using a quantum computer. Note that Shor's algorithm being a quantum algorithm does not necessarily mean that everything has to be done through use of properties of quantum physics. It just means that the algorithm exhibits some form of quantum computation part, classical parts are still present in the algorithm.

1. (classical) Pick a random integer number m where $m < N$ and both m and N are co-prime.
2. (quantum) Determine the period P of the function $f(x) = m^x \bmod N$ using a quantum computer.
3. (classical) Check if the number P is odd or even. If P is odd, go back to step 1. If P is even, proceed to step 4.
4. (classical) If $(m^{P/2} + 1) = 0 \bmod N$, the solution is trivial and we need to go back to step 1. If $(m^{P/2} + 1) \neq 0 \bmod N$, the solution is non-trivial and we can proceed to step 5.
5. (classical) We can compute l from the integer number m using $l = \gcd(m^{P/2} - 1, N)$. We found l.

2.6 Post-quantum cryptography

Suppose that there is a situation where suddenly a sufficiently large quantum computer is available. This large quantum computer, with higher computational power than a classical binary computer, would attack current classical based cryptographic algorithms. Notably, the examples of Diffie-Hellman and RSA which we described above would not hold anymore as the discrete logarithm problem and integer factorization would be computationally easier to solve as we described in Shor's algorithm Section 2.5. With post-quantum cryptography (PQC) we describe the situation where algorithms that are attacked by this quantum computer can resist attacks from it [1]. We distinguish several main classes of cryptographic essentials:

- Hash-based cryptography
- Code-based cryptography
- Lattice-based cryptography
- Multivariate-quadratic-equations cryptography
- Secret-key cryptography
- Supersingular isogeny cryptography

The above-mentioned systems are believed to withstand possible attacks from quantum computers. Each of these classes are so broad that one could write a whole book about them. In this thesis however, we will look more closely at parts of the lattice-based cryptography. As the name suggest, it describes cryptographic primitives through the use of lattices. More on this topic will be explained in Chapter 3.

Though not unrelated, note that post-quantum cryptography is different from regular quantum cryptography in the sense that quantum cryptography focuses on the usage of quantum physics to secure data and detect possible communication breaches whereas PQC describes quantum resistant crytopgrahic algorithms.

Chapter 3

Lattice-Based Cryptography

This chapter gives an overview of lattice-based cryptography along with its properties and applications. Lattice-based cryptography is considered to be immune against attacks from a sufficiently large quantum computer. That is, no algorithm have yet been found to solve the computational hardness used in lattice-based primitives in reasonable time. First, a general introduction to lattices is given. Afterwards, the cryptography part is explained.

3.1 Notations

Throughout this chapter we will be using many mathematical notations to explain topics. The notations used in this thesis are mostly the same as the general notations used on the topic of linear algebra. However, for consistency purposes we will define these notations here.

Let $\mathbb{R}^n$ be the Euclidean vector space. Vectors will be given by boldface letters with an arrow on it. The dot product for two vectors $\vec{\boldsymbol{a}}$ and $\vec{\boldsymbol{b}}$ is denoted by

$$\vec{\boldsymbol{a}} \cdot \vec{\boldsymbol{b}} = \sum_{i=1}^{n} a_i b_i \tag{3.1}$$

The norm or length of a vector $\vec{\boldsymbol{a}}$ is given by

$$\|\vec{\boldsymbol{a}}\| = \sqrt{a_1^2 + \ldots + a_n^2} \tag{3.2}$$

Any vector $\vec{a}$ will be written as a column vector or its transposed version:

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \qquad \vec{a} = (a_1, a_2, \ldots, a_n)^\top$$

with $\vec{a}$ in $\mathbb{R}^n$

Unit vectors are given by $\vec{e_1}, \ldots, \vec{e_n}$. Unit vectors always have a length of 1.

The rounding notation rounds any real variable to an integer closest to the real variable. In the case of ambiguity, the rounding function will always round up. The notation is given by

$$\lfloor . \rceil : \mathbb{R} \to \mathbb{Z}$$

The vectors $\vec{a}_1, \ldots, \vec{a}_m \in \mathbb{R}^n$ are defined to be linear independent if there exists no $x_1, \ldots, x_m \in \mathbb{R}$ with some of the $x_i \neq 0$:

$$\sum_{i=1}^{m} x_i \vec{a}_i = \vec{0} \tag{3.3}$$

If the condition does hold, the vectors are said to be linear dependent.

Matrices are denoted by a capital letter $\mathcal{A}$ with dimensions specified. Consider a matrix $\mathcal{A}$ with dimension $m \times n$. The matrix $\mathcal{A}$ is denoted by

$$\mathcal{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

One particular special matrix we will discuss is the Gram-matrix. The notation of the $m \times m$ Gram-matrix with vectors $\vec{a_1}, \ldots, \vec{a_m}$ consists of all pairwise dot products of $\vec{a_i}$. Given the matrix

$$\mathcal{A} = \begin{bmatrix} | & | & & | \\ \vec{a_1} & \vec{a_2} & \cdots & \vec{a_m} \\ | & | & & | \end{bmatrix}$$

the equivalent Gram-matrix is given by $\mathcal{A}^T\mathcal{A}$. An important property is that the determinant of the Gram-matrix is that it is always ≥ 0 and only equal to 0 if the vectors are linearly dependent.

3.2 Lattice introduction

A lattice is defined to be a set of points in a m-dimensional Euclidean space having a periodic structure. Given n linear independent vectors $\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n \in \mathbb{R}^m$ with $m \geqslant n$, the generated lattice is formally defined to be the set of vectors:

$$\mathcal{L}(\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n) = \left\{ \sum_{i=1}^{n} a^i b^i \;\middle|\; a^i \in \mathbb{Z} \right\}$$

where vectors $\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n$ are called the *basis* of the generated lattice. The integers n and m are called *rank* and *dimension* respectively. In this thesis, we will only consider lattices where the rank and dimension are equal $(m = n)$.

All integer linear combinations of the vectors $\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n$ generate a lattice. Note that the same lattice can be generated in several different ways. That is, the same lattice can be generated by different bases. Figure 3.1 gives an example of a 2-dimensional lattice with basis $\{\vec{\boldsymbol{b}}_1, \vec{\boldsymbol{b}}_2\}$. Notice how the integer linear combinations $\vec{\boldsymbol{b}}_1 + \vec{\boldsymbol{b}}_2$ and $2\vec{\boldsymbol{b}}_1 + \vec{\boldsymbol{b}}_2$ are also lattice points on the 2-dimensional plane.

Suppose we have a unimodular matrix $\mathcal{U}$, that is, an integer square matrix with determinant either $+1$ or -1. A basis $\mathcal{B}$ and a multiplied version $\mathcal{BU}$ will then generate the same lattice. In general, any square matrix $\mathcal{A}$ multiplied with a unimodular matrix $\mathcal{U}$ of the same size will always generate the same lattice $\mathcal{L}$. The determinant of a lattice $\mathcal{L}$ is computed by taking the absolute value of the determinant of the matrix basis $\mathcal{B}$. Thus, $\det(\mathcal{L}(\mathcal{B})) = |\det(\mathcal{B})|$ [1].

We can represent the lattice basis vectors $\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n \in \mathbb{R}^n$ as a matrix $\mathcal{B}$ where

$$\mathcal{B} = [\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n] \in \mathbb{R}^n.$$

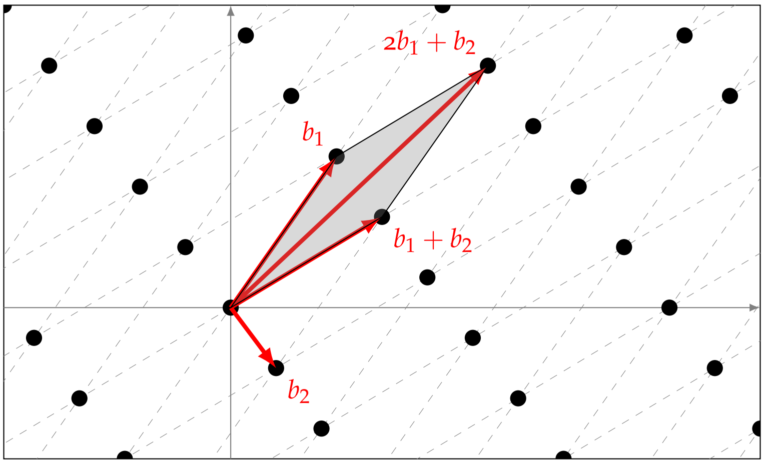

Figure 3.1: A 2-dimensional Lattice with basis $\vec{\boldsymbol{b}}_1$ and $\vec{\boldsymbol{b}}_2$.

The bases do not necessarily have to be unique to span the same lattice $\mathcal{L}$. As a second example, let us look at the lattice $\mathbb{Z}^n$. If we take a subsection of the lattice $\mathcal{L}$ centered at the origin (numerical value 0) we can intuitively already think of simple bases that span the lattice, namely the unit vectors. Consider a lattice plane, spanned by the unit vectors $\vec{e_1}$ and $\vec{e_2}$. Now consider another lattice plane spanned by vectors $\vec{b_1} = (1,1)^\top$ and $\vec{b_2} = (2,1)^\top$. See Figure 3.2 for a comparison between both lattice planes. It is easy to see that the same lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ is generated by $\{\vec{e_1} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \vec{e_2} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\}$ but also by $\{\vec{b_1} = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right], \vec{b_2} = \left[\begin{smallmatrix}2\\1\end{smallmatrix}\right]\}$.

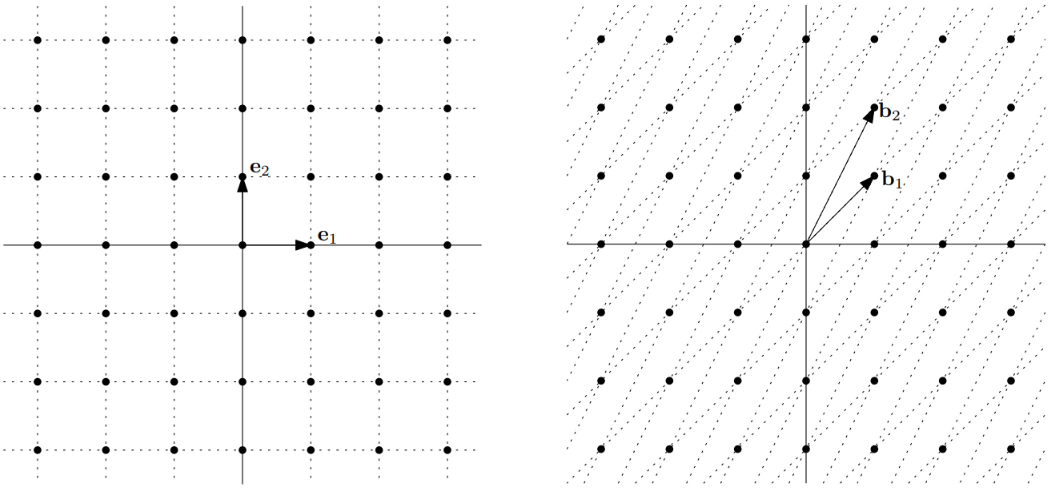

Figure 3.2: Lattice $\mathbb{Z}^n$ spanned by basis $\vec{e_1}$ and $\vec{e_2}$ (left) and basis $\vec{b_1} = (1,1)^\top$ and $\vec{b_2} = (2,1)^\top$ (right) — example taken from [3].

3.3 Shortest vector problem

One particular interesting problem when working with lattices is the shortest vector problem (SVP). Informally, the shortest vector problem describes a non-zero vector in a lattice generated by a certain basis $\vec{b_1}, \ldots, \vec{b_n} \in \mathbb{R}^n$ which is closest to the origin of the basis vectors.

A more formal definition is as follows. Given a certain basis $\vec{b_1}, \ldots, \vec{b_n} \in \mathbb{R}^n$ generating a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$, find a non-zero vector $\vec{x} \in \mathcal{L}$ such that

$$\|x\| = \min_{\vec{v} \in \mathcal{L} \setminus \{0\}} \|v\|$$

Often, the length of a shortest vector is denoted by λ, resulting in $\lambda = \|x\|$. A two-dimensional schematic impression is given in Figure 3.3. Here we see that a circle is drawn from the origin of the two basis vector $\vec{b_1}$ and $\vec{b_2}$ and that $\vec{b_2}$ is the closest vector. Here, we see that for the norm it holds that $\lambda = \|\vec{b_2}\|$. When talking about the norm, we define it to be the Euclidean norm as described in equation 3.2 thus being $||\,\vec{b}\,|| = \sqrt{\sum_i b_i^2}$.

Though graphically clear and relatively easy to solve in a 2-dimensional situation as proven by Gauss [13], this problem can be upscaled to a n-dimensional situation. In the n-dimensional case the problem becomes harder to solve. So hard that in 1996 Ajtai proved that the shortest vector problem is NP-hard for randomized reductions [14, 15].

An exact solution for the SVP takes a lot of computational power. Even an approximation to factors of poly(n) takes up to $2^{\mathcal{O}(n)}$ time [16].

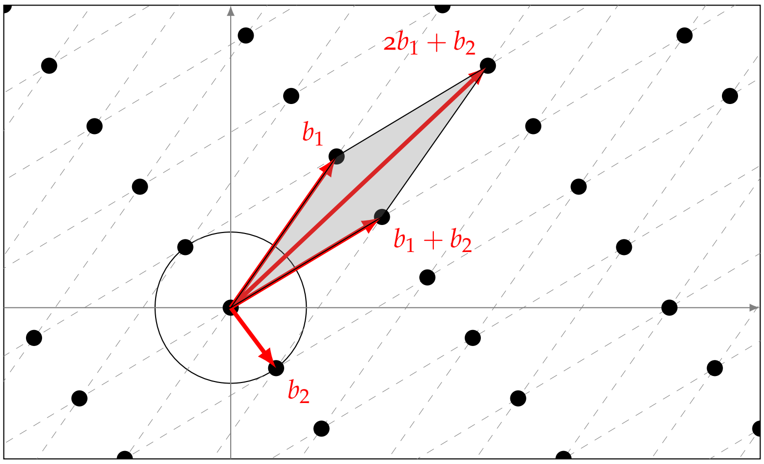

Figure 3.3: A 2-dimensional lattice with basis $\vec{b}_1$ and $\vec{b}_2$ and a schematic representation of the shortest vector $\vec{b}_2$.

3.3.1 Approximate shortest vector problem

A variant of the shortest vector problem is the α-approximate shortest vector problem. Essentially, it describes that when finding the shortest vector in a lattice we want to find the instance of that vector that is at most α times the length of the shortest vector. Schematically, we again draw a circle from the origin of the basis, but this time we scale the radius up with a factor α. We denote the length of it by $\alpha\lambda$. Essentially, we are looking for a vector with a maximum length of $\alpha\lambda$. This vector is very likely to be not exactly on the edge of the drawn circle, but be contained in it, hence we approximate this lattice point and thus approximate the length of the shortest vector [17].

Formally, given a certain basis $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^n$ generating a lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ and an approximation factor α with $\alpha \geq 1$. Find the non-zero vector vector $\vec{x} \in \mathbb{L}$ such that

$$\|x\| = \alpha \times \min_{\vec{v} \in \mathcal{L} \setminus \{0\}} \|v\|$$

3.4 Closest vector problem

The shortest vector problem explained in Subsection 3.3 is essentially a specific case of the closest vector problem (CVP). That is, the CVP is a generalization of the SVP. Informally, the CVP, just as the SVP, requires a basis $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^n$ as well as another point $\vec{p}$ which can be a vector in the space which is not necessarily part of the lattice. Given this input, the basis will generate a lattice and have a (vector)point $\vec{p}$ somewhere in the space of this lattice. The closest vector problem then describes the lattice point generated by the basis which is closest to the arbitrary (vector)point $\vec{p}$.

Formally, given a certain basis $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^n$ generating a lattice $\mathcal{L}$ and a target vector $\vec{p}$. Find a lattice-vector $\vec{x} \in \mathcal{L}$ such that $\|\vec{p} - \vec{x}\| = \text{dist}(\vec{p}, \mathcal{L})$.

Figure 3.4 gives an example of a generated lattice with an arbitrary (vector)point $\vec{p}$ and closest lattice point z.

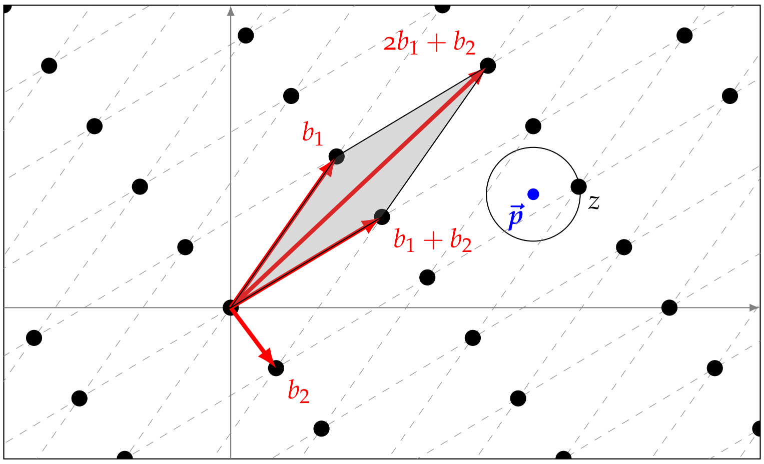

Figure 3.4: A 2-dimensional lattice with bases $\vec{b_1}$ and $\vec{b_2}$ and a schematic representation of the closest vector to point $\vec{p}$.

It can be noted that, essentially, the closest vector problem is the same as the shortest vector problem given bases which span the same lattice and an arbitrary point $\vec{p}$ which is *closest* to the origin on the spanned basis. The fact that one problem is perhaps intuitively an easier variant of the other problem is no reason to suspect one is easier to compute than the other. As a matter of fact, in 1999 Goldreich et al. proved that both the SVP and the CVP contain the same level of hardness under randomized reductions [18].

3.4.1 Approximate closest vector problem

As with the approximate version of the shortest vector problem, there is also an approximate version of the closest vector problem. In accordance with the methods above, given a certain basis $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^n$ generating a lattice $\mathcal{L}$, a target vector $\vec{p}$ and an approximation factor α with $\alpha \geq 1$. Find the lattice-vector $\vec{x} \in \mathcal{L}$ such that $\|\vec{p} - \vec{x}\| \leqslant \alpha \times \operatorname{dist}(\vec{p}, \mathcal{L})$.

3.5 Computational hardness of lattice-based cryptography

As explained in Chapter 2 one-way functions are one of the many cryptographic primitives. In 1996, Ajtai published a paper describing a way to get one-way functions from lattices. This one-way function is based on the hardness of the shortest vector problem.

One ground breaking and a unique feature of lattice-based cryptography is that worst-case hardness is equal to average-case hardness for certain lattice problems as Ajtai proved in 1996 [14, 19].

3.6 Lattice reduction

Lattice basis reduction is one very important part in lattice-based cryptopgraphy. Originally, lattice basis reduction was used to attack other forms of cryptography, but soon it was discovered that cryptographic primitives could also be constructed from lattices. In 1982, Lenstra, Lenstra and Lovász introduced an algorithm called the LLL-algorithm. It reduces the basis of a given lattice [20]. Another useful feature this algorithm produces is an approximation of the shortest vector in the lattice, essentially partly solving the shortest vector problem described in Section 3.3. Though originally not specifically mentioned to be of use in lattices, their work turned out to be very valuable in this cryptography part, mainly because the algorithm for finding a reduced lattice basis runs in polynomial time [3].

3.6.1 Gram-Schmidt Orthogonalization

An algorithm that is used in the LLL-algorithm is the Gram-Schmidt Orthogonalization process. It is not specifically classified as a lattice reduction algorithm, but it comes very close to it. Small changes to the original algorithm make it useful for finding a reduced lattice basis.

We define the Gram-Schmidt reduction of basis $\vec{\boldsymbol{b}}_1, \vec{\boldsymbol{b}}_2, \ldots, \vec{\boldsymbol{b}}_n$ of a certain lattice $\mathcal{L}$ to be

$$\vec{\boldsymbol{b}}_i^* = \vec{\boldsymbol{b}}_i - \sum_{j=1}^{i-1} \mu_{ij} \vec{\boldsymbol{b}}_j^* \tag{3.4}$$

where we define the GSO coëfficient μ to be

$$\mu_{ij} = \frac{\vec{\boldsymbol{b}}_i \cdot \vec{\boldsymbol{b}}_j^*}{\vec{\boldsymbol{b}}_j^* \cdot \vec{\boldsymbol{b}}_j^*} \qquad \text{with } 1 \leq j < i \leq n \tag{3.5}$$

Note that this graphically implies the projection of the vector $\vec{\boldsymbol{b}}_i$ onto $\vec{\boldsymbol{b}}_j^*$.

To orthonormalize the reduced basis we divide all vectors $\vec{\boldsymbol{b}}_i^*$ by their own length. See Figure 3.5 for an overview of a Gram-Schmidt Orthogonalization process with basis $\vec{\boldsymbol{b}}_1 = (1,2)^\top$ and basis $\vec{\boldsymbol{b}}_2 = (1,0)^\top$. Furthermore, Figure 3.5 also gives an example of the normalization on the basis. The Gram-Schmidt algorithm is given in the box below.

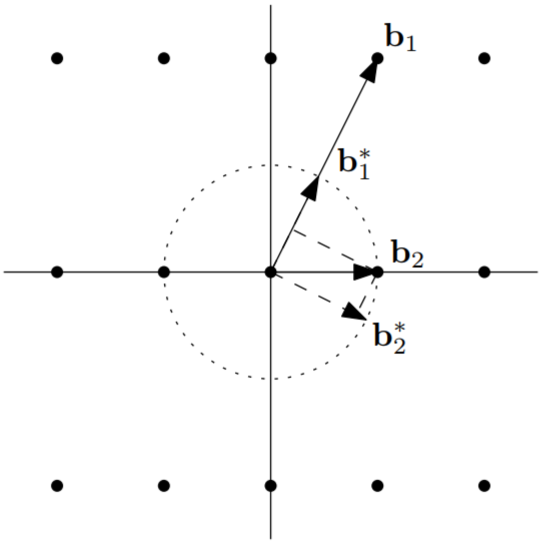

Figure 3.5: Gram-Schmidt Orthogonalization process with normalization for $\vec{b}_1 = (1,2)^\top$ and $\vec{b}_2 = (1,0)^\top$ — example taken from [3].

Gram-Schmidt algorithm — example taken from [3]

```
input  : $(\vec{b}_1, \dots, \vec{b}_n) \in \mathbb{R}^n$
output : $(\vec{b}_1^*, \dots, \vec{b}_n^*) \in \mathbb{R}^n$
$\vec{b}_1^* = \vec{b}_1$
for $i = 2$ to $n$ do
    $\vec{v} = \vec{b}_i$
    for $j := i - 1$ downto 1 do
        $\mu_{ij} = \frac{\vec{b}_i \cdot \vec{b}_j^*}{\vec{b}_j^* \cdot \vec{b}_j^*}$
        $\vec{v} = \vec{v} - \mu_{ij} \vec{b}_j^*$
    end
    $\vec{b}_i^* = \vec{v}$
end
return $(\vec{b}_1^*, \dots, \vec{b}_n^*)$
```

3.7 LLL-algorithm

As stated before, the LLL-algorithm (sometimes called L3) is an efficient method of finding a near-to-suitable orthogonal basis of a lattice. The LLL-algorithm solves the SVP in polynomial time reaching an approximation-factor of $2^{\mathcal{O}(n)}$ with n being the dimension of the lattice. Also, the LLL-algorithms is used as an attack algorithm on current cryptosystems based on the knapsack problem and also several cases of RSA [1]. In general, the execution of the LLL-algorithm is divided into two parts.

- Reduction of the given (often not a basis) working vector, essentially performing size reduction.

- Swapping of the working vector and basis vector based on whether or not the Lovász condition is satisfied. Informally, the Lovász condition checks if the chosen working vector is sufficiently large to be the new basis vector.

The starting input of the LLL-algorithm are the current basis vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n$. In total, there are three inputs for the LLL-algorithm:

- the current basis vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n$
- the Gram-Schmidt Orthogonalization reduced basis vectors $\vec{b}_1^*, \vec{b}_2^*, \ldots, \vec{b}_n^*$
- a constant k which denotes the amount of working basis vectors which are to be reduced

The box below gives an overview of the Lenstra-Lenstra-Lovász algorithm as described in their paper. Note how in the algorithm, the previously mentioned Gram-Schmidt orthogonalization process is used.

LLL algorithm with Euclidean norm $\delta = 3/4$ — example taken from [3]

```
input  : $(\vec{b}_1, \ldots, \vec{b}_n) \in \mathbb{Z}^n$
output : LLL reduced basis $(\vec{b}_1, \ldots, \vec{b}_n)$
Compute Gram-Schmidt basis $(\vec{b}_1^*, \ldots, \vec{b}_n^*)$ and coefficients $\mu_{ij}$ for $1 \leq j < i \leq n$
Compute $\vec{b}_i^* \cdot \vec{b}_i^*$ for $1 \leq i \leq n$
$k = 2$
while $k \leq n$ do
    for $j = k - 1$ downto 1 do
        $\vec{b}_k = \vec{b}_k - \lfloor \mu_{k,j} \rceil \vec{b}_j$
        Update values $\lfloor \mu_{k,j} \rceil$ for $1 \leq j < k$
    end
    if $\mathcal{B}_k \geq (\delta - \mu_{k,k-1}^2)\mathcal{B}_{k-1}$ then
        $k = k + 1$
    else
        Swap $\vec{b}_k$ with $\vec{b_{k-1}}$
        Update values $\vec{b}_k^*, \vec{b}_{k-1}^*, \mathcal{B}_k, \mathcal{B}_{k-1}, \mu_{k-1,j}$ and $\mu_{k,j}$ for $1 \leq j < k$, and $\mu_{i,k}, \mu_{i,k-1}$ for $k < i \leq n$
        $k = \max\{2, k - 1\}$
    end
end
return $(\vec{b}_1^*, \ldots, \vec{b}_n^*)$
```

Suppose the basis vectors which we are working with are defined to be $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{k-1}, \vec{b}_k$ and our goal is to reduce the vector $\vec{b}_k$. The first step is to subtract any integer multiples of the vectors $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{k-1}$

So, if given only the vectors $\vec{b}_{k-1}$ and $\vec{b}_k$, we first need to subtract an integer multiple of $\vec{b}_k$ from $\vec{b}_{k-1}$. This means we need to swap the vectors $\vec{b}_{k-1}$ and $\vec{b}_k$. Given that we have a new vector $\vec{b}_{k-1}$, we need to redo the process, but this time with $\vec{b}_{k-1}$ as our new working vector.

The swapping of the two vectors is determined by the Lovász condition. It is a condition that checks whether or not the size of the working vector is sufficient.

Let $\vec{\boldsymbol{b}}_{\boldsymbol{k}}$ be our working vector. Let again the parameter $\mu_{k,k-1}$ be defined as

$$\mu_{k,k-1} = \frac{\vec{\boldsymbol{b}}_{\boldsymbol{k}} \cdot \vec{\boldsymbol{b}}^*_{\boldsymbol{k-1}}}{\vec{\boldsymbol{b}}^*_{\boldsymbol{k-1}} \cdot \vec{\boldsymbol{b}}^*_{\boldsymbol{k-1}}}$$

Note that this is the same equation 3.5 as described in the Gram-Schmidt Orthogonalization. Important to note here is that when determining the new working vector, we use the *rounded* GSO-coefficient so we maintain integer coordinates. This slightly changes the basis vectors but not their projections [3].

We say that the Lovász condition is satisfied if it holds that

$$||\vec{\boldsymbol{b}}^*_{\boldsymbol{k}}||^2 \geq \left(\frac{3}{4} - \mu^2_{k,k-1}\right) ||\vec{\boldsymbol{b}}^*_{\boldsymbol{k-1}}||^2 \tag{3.6}$$

If the condition is satisfied we can conclude that $\vec{\boldsymbol{b}}_{\boldsymbol{k}}$ is an acceptable basis vector and we can proceed to the vector $\vec{\boldsymbol{b}}_{\boldsymbol{k+1}}$ as the new working vector. If the condition is not satisfied however, we swap the vectors and make $\vec{\boldsymbol{b}}_{\boldsymbol{k-1}}$ the new working vector.

3.7.1 Example

In this example we apply the LLL-algorithm to certain two-dimensional lattice $\mathcal{L}$ whose basis is spanned by the vectors $\vec{\boldsymbol{b}}_{\boldsymbol{1}} = \begin{bmatrix} 180 \\ 15 \end{bmatrix}$ and $\vec{\boldsymbol{b}}_{\boldsymbol{2}} = \begin{bmatrix} 312 \\ 80 \end{bmatrix}$.
We start by choosing one of the vectors as our first basis vector. Next, we use this vector to reduce the second vector to create a new candidate basis vector. We conveniently choose $\vec{\boldsymbol{b}}_{\boldsymbol{1}}$ as our first basis vector and choose $\vec{\boldsymbol{b}}_{\boldsymbol{2}}$ as the vector we are reducing.

We start off by doing the Gram-Schmidt reduction using $\vec{\boldsymbol{b}}^*_{\boldsymbol{1}} = \vec{\boldsymbol{b}}_{\boldsymbol{1}}$ as our first Gram-Schmidt vector to find the Gram-Schmidt orthogonal basis. Recall the formula from Subsection 3.6.1. Our first Gram-Schmidt vector is: $\vec{\boldsymbol{b}}^*_{\boldsymbol{1}} = \begin{bmatrix} 180 \\ 15 \end{bmatrix}$ and we reduce the second vector $\vec{\boldsymbol{b}}_{\boldsymbol{2}}$.

We compute the $\vec{\boldsymbol{b}}_{\boldsymbol{2}}$ by the Gram-Schmidt method $\vec{\boldsymbol{b}}^*_{\boldsymbol{2}} = \vec{\boldsymbol{b}}_{\boldsymbol{2}} - \frac{\vec{\boldsymbol{b}}_{\boldsymbol{2}} \cdot \vec{\boldsymbol{b}}^*_{\boldsymbol{1}}}{\vec{\boldsymbol{b}}^*_{\boldsymbol{1}} \cdot \vec{\boldsymbol{b}}^*_{\boldsymbol{1}}} * \vec{\boldsymbol{b}}^*_{\boldsymbol{1}}$ as seen in equation 3.4. Filling in the numbers results in:

$$\vec{\boldsymbol{b}}^*_{\boldsymbol{2}} = \begin{bmatrix} 312 \\ 80 \end{bmatrix} - \frac{\begin{bmatrix} 312 \\ 80 \end{bmatrix} \cdot \begin{bmatrix} 180 \\ 15 \end{bmatrix}}{\begin{bmatrix} 180 \\ 15 \end{bmatrix} \cdot \begin{bmatrix} 180 \\ 15 \end{bmatrix}} * \begin{bmatrix} 180 \\ 15 \end{bmatrix} \text{ which results in } \vec{\boldsymbol{b}}^*_{\boldsymbol{2}} \approx \begin{bmatrix} -4,47 \\ 53,63 \end{bmatrix}$$

Note that we graphically computed the second vector $\vec{b}_2$ minus the *projection* onto the first vector $\vec{b}_1$. Also note that the GSO-coëfficient

$$\mu = \frac{\begin{bmatrix}312\\80\end{bmatrix} \cdot \begin{bmatrix}180\\15\end{bmatrix}}{\begin{bmatrix}180\\15\end{bmatrix} \cdot \begin{bmatrix}180\\15\end{bmatrix}} \approx 1,758.$$

To compute our new basis vector $\vec{b}_2$, we take the *rounded* GSO-coefficient to multiply with the Gram-Schmidt vector resulting in the following formula

$$\vec{b}_{2_{new}} = \vec{b}_2 - \lfloor \mu \rceil \, \vec{b}_1^* \tag{3.7}$$

where $\lfloor \mu \rceil$ is the rounded μ coefficient. Inserting the numbers gives $\lfloor \mu \rceil = 2$ and $\vec{b}_{2_{new}} = \begin{bmatrix}-48\\50\end{bmatrix}$.

So far we have computed $\vec{b}_2^*$ and $\vec{b}_{2_{new}}$ from the assumption $\vec{b}_1^* = \vec{b}_1$. We can now check with the Lovász condition 3.6 whether our candidate basis is good or we should switch the basis vectors. For the Lovaśz condition to be checked we need to compute $||\vec{b}_1^*||^2$ and $||\vec{b}_2^*||^2$. Also, we need to compute μ^2 for $\vec{b}_{2_{new}}, \vec{b}_1^*$. We find that

$$||\vec{b}_1^*||^2 = 32625 \qquad ||\vec{b}_2^*||^2 \approx 2895,89 \qquad \frac{3}{4} - \mu^2 \approx 0,69$$

From equation 3.6 we can easily see that the Lovász condition does not hold as $2895,89 \not\geqslant \frac{3}{4} - (-0,24)^2 * 32625$. This means our candidate basis was not good and we need to swap our vectors $\vec{b}_1$ and $\vec{b}_{2_{new}}$ and redo the process. Swapping the vectors results in our new vectors $\vec{b}_1 = \begin{bmatrix}-48\\50\end{bmatrix}$ and $\vec{b}_2 = \begin{bmatrix}180\\15\end{bmatrix}$.

Again, we start off by doing the Gram-Schmidt reduction using $\vec{b}_1^* = \vec{b}_1$ as our first Gram-Schmidt vector. And we compute $\vec{b}_2$ using the Gram-Schmidt method 3.4 to find $\vec{b}_2^* = \begin{bmatrix}101,17\\97,12\end{bmatrix}$. Using the rounded GSO-coëfficient equation 3.7 we find the vector $\vec{b}_{2_{new}}$ to be $\vec{b}_{2_{new}} = \begin{bmatrix}84\\115\end{bmatrix}$.

We now want to check if the Lovász condition holds for our new vectors. Again we compute $||\vec{b}_1^*||^2$ and $||\vec{b}_2^*||^2$ and μ^2 for $\vec{b}_{2_{new}}$ and $\vec{b}_1^*$.

$$||\vec{b}_1^*||^2 = 4804 \qquad ||\vec{b}_2^*||^2 \approx 19666,60 \qquad \frac{3}{4} - \mu^2 \approx 0,62 \quad (\mu \approx 0,358)$$

From equation 3.6 we see that the Lovász condition holds as $19666,60 \geqslant \frac{3}{4} - (0,358)^2 * 4804$. Now we can move to the vector $\vec{b}_{k+1}$. However, since we only have two vectors our process is finished and we can conclude that the vectors $\vec{b}_1 = \begin{bmatrix}-48\\50\end{bmatrix}$ and $\vec{b}_2 = \begin{bmatrix}180\\15\end{bmatrix}$ are our near-to-suitable orthogonal set of basis vector for the lattice $\mathcal{L}$.

3.8 Babai's algorithm

In 1986 Laśzló Babai introduced two algorithms that could solve the approximate closest vector problem in reasonable time [21]. The essence of Babai's algorithm is that is works efficient when working with a good basis $\mathcal{B}$ but not so efficient when workting with a bad basis $\mathcal{B}$. We define a good basis $\mathcal{B}$ to be a basis where the basis vector are close to orthogonal to each other. We define a bad basis to be a basis where the basis vectors are close to parallel to each other.

The two algorithms, the rounding method and the nearest-plane method both use the Lenstra-Lenstra-Lovász algorithm to find a reduced lattice basis. Of the two method, the nearest-plane algorithms gives a better approximation than the rounding method at the cost of extra computation [3]. The rounding method is however, reasonably good and cheap in sense of computational power that we will discuss only this method here.

Suppose there is a lattice $\mathcal{L}$ with basis $\mathcal{B}$. Let there also be a target vector $\vec{z} \in \text{span}(\mathcal{L})$. Looking for a rounded vector $\vec{v}$, the rounding method describes finding this vector $\vec{v}$ as follows:

$$\mathcal{B}^{-1}(\vec{z}) = \sum_{i=1}^{n} \lambda_i \vec{\boldsymbol{b}}_{\boldsymbol{i}} \tag{3.8}$$

The part $\lambda_i \vec{\boldsymbol{b}}_{\boldsymbol{i}}$ describes a unique linear combination from basis vectors $\vec{\boldsymbol{b}}_{\boldsymbol{1}}, \ldots, \vec{\boldsymbol{b}}_{\boldsymbol{n}}$ and $\lambda_i \in \mathbb{R}$. The vector $\vec{z}$ in accordance with basis $\mathcal{B}$ is given by $\mathcal{B}^{-1} \cdot \vec{z}$ resulting in $\lambda_1, \ldots, \lambda_n = \mathcal{B}^{-1}\vec{z}$. Babai's rounding method now describes the rounding part. That is, it rounds each instance λ_i to the closest integer in $\mathbb{Z}$ denoted by $\lfloor \lambda_i \rceil$.

$$\left\lfloor \mathcal{B}^{-1}\vec{z} \right\rceil = \sum_{i=1}^{n} \lfloor \lambda_i \rceil \vec{\boldsymbol{b}}_{\boldsymbol{i}} \tag{3.9}$$

The vector $\vec{v}$ can now be computed by taking the rounded $\lfloor \lambda_i \rceil$ as coefficients of the linear combination of the elements of $\vec{\boldsymbol{b}}_{\boldsymbol{i}}$. This results in our lattice vector $\vec{v}$:

$$\vec{v} = \mathcal{B} \left\lfloor \mathcal{B}^{-1}\vec{z} \right\rceil \tag{3.10}$$

Note that Babai's rounding algorithm does not necessarily always give the right answer when using a bad basis. To show this claim, let us define a lattice $\mathcal{L}$ which is spanned by $\mathbb{Z}^2$, much like the lattice seen on the left side in Figure 3.2. Let us take the bases $\vec{\boldsymbol{b}}_{\boldsymbol{1}} = (1,1)^\top$ and $\vec{\boldsymbol{b}}_{\boldsymbol{2}} = (1,0)^\top$ and set our target-vector $\vec{z} = (\frac{13}{8}, \frac{1}{4})^\top$. Determining the vector $\vec{z}$ as elements of basis $\vec{\boldsymbol{b}}$ gives $\vec{z} = \frac{1}{4}\vec{\boldsymbol{b}}_{\boldsymbol{1}} + \frac{11}{8}\vec{\boldsymbol{b}}_{\boldsymbol{2}}$. Rounding the coordinates $\mathcal{B}^{-1}\vec{z} = (\frac{1}{4}, \frac{11}{8})^\top$ gives $\mathcal{B}^{-1}\vec{z} = (0,1)^\top$. Expressed as factors of the basis $\vec{\boldsymbol{b}}$ gives $0\vec{\boldsymbol{b}}_{\boldsymbol{1}} + 1\vec{\boldsymbol{b}}_{\boldsymbol{2}} = (1,0)^\top$. However, when studying the lattice we find that the closest point to the targetvector $\vec{z} = (2,0)^\top$ which is $2\vec{\boldsymbol{b}}_{\boldsymbol{2}}$. Thus, we see that Babai's algorithm does not always right us the right answer [3]. Figure 3.6 gives a graphical overview of the situation above.

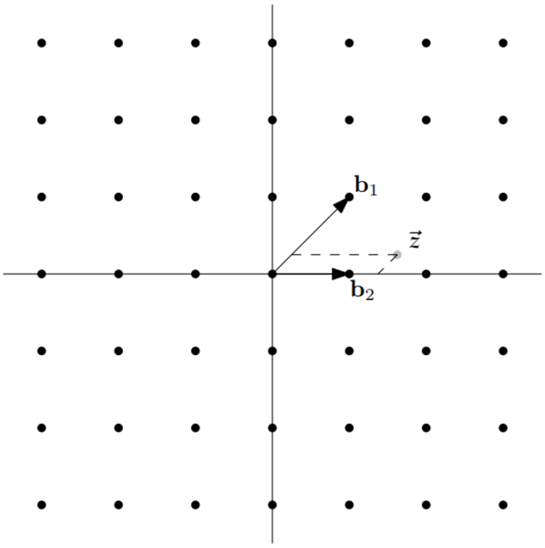

Figure 3.6: Overview of Babai's rounding method with errornous instance — example taken from [3].

3.9 Goldreich-Goldwasser-Halevi cryptosystem

One important cryptosystem in lattice-based cryptography is the Goldreich-Goldwasser-Halevi (GGH) cryptosystem. Named after its inventors, the cryptosystem was published in 1997 and later refined by Nguyen in 1999 [22]. The GGH cryptosystem was the first lattice-based cryptosystem that was specifically defined to be using lattices. The GGH cryptosystem relies on the hardness of finding the closest vector in a lattice as described in Section 3.4. To explain the setup of the GGH cryptosystem we use the same basic person system as introduced in Figure 2.2.

3.9.1 Setup

Similar to our other examples, we consider Alice to be the receiver and Bob to be the sender. The setup of the GGH cryptosystem consists of several prerequisites:

- Alice chooses a basis $\mathcal{B}$ consisting of an amount n near-to-suitable orthogonal vectors. Alice keeps this basis a secret.
- Alice also chooses another basis $\mathcal{B}'$ consisting of an amount n near-to-parallel vectors. This basis is made public.
- Bob determines his lattice message $\vec{m}_{Lattice}$ which is a linear combination of $\mathcal{B}'$ computed by his message $\vec{m}$ with values $m_1, \ldots, m_n$. Thus, $\vec{m}_{Lattice} = \mathcal{B}'\vec{m}$.

- Bob chooses a random error vector $\vec{e}$ and determines a ciphertext $\vec{c} = \vec{m}_{Lattice} + \vec{e}$. This ciphertext $\vec{c}$ is then sent to Alice over a public channel.
- Alice solves the closest vector problem for the ciphertext $\vec{c}$ using the private basis $\mathcal{B}$ to find the message $\vec{m}$.

If eavesdropper Eve wants to find the secret message $\vec{m}$ the adversary has to solve the closest vector problem using the public basis $\mathcal{B}'$. Because this basis consists of only nearly parallel basis vectors, solving the closest vector problem in a reasonable amount of time is very unlikely [23].

The original creators of the GGH algorithms also introduced a certain security parameter σ. The chosen error vector $\vec{e}$ is randomly generated from the set $\{-\sigma, \sigma\}^n$ where n denotes the dimension of the lattice [3].

3.9.2 Vulnerability

Adding noise to a message introduces a important vulnerability. If Bob hypothetically adds a noise parameter $\vec{e}$ to his message that coincidentally shifts the computed lattice point to a nearest lattice point it could breach the security of the lattice [22]. Hence, it is important that a good error vector $\vec{e}$ and good basis vectors are chosen [23].

3.9.3 Encryption using GGH

Encrypting the message $\vec{m}$ Bob takes the linear combination of the public basis $\mathcal{B}'$ with the message values $m_1, \ldots, m_n$ to receive a ciphertext $\vec{c}$. Essentially, the computation is

$$\vec{c} = \mathcal{B}'\vec{m} + \vec{e} \tag{3.11}$$

3.9.4 Decryption using GGH

To receive the original message $\vec{m}$ we need to use the inverse of matrices $\mathcal{B}$ and $\mathcal{B}'$. Essentially, the computation for decrypting the ciphertext $\vec{c}$ is

$$\vec{m} = \mathcal{B}'^{-1}\mathcal{B} \left\lfloor \mathcal{B}^{-1}\vec{c} \right\rceil \tag{3.12}$$

We can see that we actually compute $\vec{m} = \mathcal{B}'^{-1}\vec{m}_{Lattice}$ with the rounding part eliminating error $\vec{e}$.

Another way to retrieve our original message $\vec{m}$ is given below. Applying some linear algebra, we can see that:

$$\mathcal{B}^{-1}\vec{c} = \mathcal{B}^{-1}(\mathcal{B}'\vec{m} + \vec{e})$$

which can also be written as

$$= \mathcal{B}^{-1}\mathcal{B}'\vec{m} + \mathcal{B}^{-1}\vec{e}$$

Also,

$$= \mathcal{B}^{-1}\mathcal{B}\mathcal{U}\vec{m} + \mathcal{B}^{-1}\vec{e}$$

Note that $\mathcal{B}^{-1}\mathcal{B}$ is the identity matrix, which is then multiplied with the unimodular matrix $\mathcal{U}$. Since it is a matrix multiplication we can omit the identity matrix resulting in:

$$\mathcal{U}\vec{m} + \mathcal{B}^{-1}\vec{e}$$

We use Babai's rounding algorithm to remove the error factor $\mathcal{B}^{-1}\vec{e}$. Applying this, results in

$$\mathcal{U}\vec{m} = \left\lfloor \mathcal{B}^{-1}\vec{c} \right\rceil$$

from which we can then determine the original message $\vec{m}$ by simply calculating:

$$\vec{m} = \mathcal{U}^{-1} \left\lfloor \mathcal{B}^{-1}\vec{c} \right\rceil$$

Note that for Alice and Bob both the encryption and decryption are easy-to-compute tasks. That is, there exist many ways to efficiently compute the product of large matrices. To decipher the message $\vec{m}$ the closest vector problem must be solved. This can be done using Babai's rounded method as described in Section 3.8. Solving the CVP with Babai's rounded method using the private basis $\mathcal{B}$ which consists of near-to-suitable orthogonal vectors is considered to be easy. Solving the CVP with the same method using the *public* basis, which consists of only near-to-parallel vectors, is considered to be very hard. The proof of this can be found in [21].

3.9.5 Example

Suppose Alice has a private matrix $\mathcal{B}$ and a public matrix $\mathcal{B}'$, both 4-dimensional.

$$\text{private}\,\mathcal{B} = \begin{bmatrix} 3 & 1 & -1 & 3 \\ 3 & -4 & 3 & 1 \\ 3 & -3 & -4 & -3 \\ -2 & -3 & -2 & 3 \end{bmatrix} \qquad \text{public}\,\mathcal{B}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -349 & 311 & 321 & 851 \end{bmatrix}$$

Given that $\mathcal{B}^{-1}\mathcal{B}' = \mathcal{U}$, we can compute the unimodular matrix $\mathcal{U}$ by:

$$\mathcal{U} = \begin{bmatrix} 0.14 & 0.063 & 0.062 & -0.10 \\ 0.09 & -0.11 & -0.05 & -0.11 \\ -0.07 & 0.12 & -0.10 & -0.08 \\ 0.14 & 0.01 & -0.08 & 0.11 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -349 & 311 & 321 & 851 \end{bmatrix} = \begin{bmatrix} 35 & -31 & -32 & -85 \\ 37 & -33 & -34 & -90 \\ 27 & -24 & -25 & -66 \\ -38 & 34 & 35 & 93 \end{bmatrix}$$

Note that we can check that this matrix is unimodular by calculating its determinant. As described in Section 3.2, the determinant of a unimodular matrix $\det(\mathcal{U}) = \pm 1$. In our case, $\det(\mathcal{U}) = 1$.

Suppose, Bob wants to encrypt a certain message $\vec{m} = \begin{bmatrix} -1 \\ 0 \\ -3 \\ 2 \end{bmatrix}$ and send this to Alice. As seen in equation 3.11 Bob has to add a certain error $\vec{e}$ to get a encrypted ciphertext $\vec{c}$.

Bob decides to choose $\vec{e} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$ as his error vector. Now bob can determine his ciphertext using equation 3.11

$$\vec{c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -349 & 311 & 321 & 851 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ -3 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ -2 \\ 1087 \end{bmatrix}$$

and send this encrypted message $\vec{c}$ to Alice.

To decrypt the encrypted message $\vec{c}$ Alice uses Babai's rounding method algorithm as described in section 3.8 to find the approximate closest vector in the lattice. She computes

$$\vec{m} = \mathcal{U}^{-1} \left\lfloor \mathcal{B}^{-1} \vec{c} \right\rceil$$

Calculating $\mathcal{B}^{-1}\vec{c}$, Alice gets

$$\mathcal{B}^{-1}\vec{c} = \begin{bmatrix} 0.14 & 0.063 & 0.062 & -0.10 \\ 0.09 & -0.11 & -0.05 & -0.11 \\ -0.07 & 0.12 & -0.10 & -0.08 \\ 0.14 & 0.01 & -0.08 & 0.11 \end{bmatrix} \begin{bmatrix} -2 \\ 1 \\ -2 \\ 1087 \end{bmatrix} = \begin{bmatrix} -108,915 \\ -115,146 \\ -83,84 \\ 118,684 \end{bmatrix}$$

To eliminate Bob's error $\vec{e}$ we round $\mathcal{B}^{-1}\vec{c}$ resulting in

$$\left\lfloor \mathcal{B}^{-1} \vec{c} \right\rceil = \begin{bmatrix} -109 \\ -115 \\ -84 \\ 119 \end{bmatrix}$$

Alice can now determine the original message $\vec{m}$ by using the unimodular matrix $\mathcal{U}$. She computes

$$\vec{m} = \mathcal{U}^{-1} \left\lfloor \mathcal{B}^{-1}\vec{c} \right\rceil = \begin{bmatrix} 3 & 1 & -1 & 3 \\ 3 & -4 & 3 & 1 \\ 3 & -3 & -4 & -3 \\ -1 & 3 & 0 & 2 \end{bmatrix} \begin{bmatrix} -109 \\ -115 \\ -84 \\ 119 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -3 \\ 2 \end{bmatrix}.$$

We see that we have retrieved the original message $\vec{m}$ which Bob's sent.

Note that an adversary Eve has only access to the public basis $\mathcal{B}'$ and thus trying to get the message $\vec{m}$ results in

$$\vec{m}_{Eve} = \left\lfloor \mathcal{B}'^{-1}\vec{c} \right\rceil = \begin{bmatrix} -2 \\ 1 \\ -2 \\ 1 \end{bmatrix}$$

which is not equal to $\vec{m}$.

Chapter 4

Conclusions

In Chapter 2 we have seen an introduction to classical public-key cryptography. Here, we explained the current state of classical cryptography and how it is currently being used. We have seen examples of a secure communication channel using the Diffie-Hellman key-exchange and also RSA, where we have been encrypting certain readable messages to ciphertext messages and sending it over a public channel to the respective receiver who can then decrypt these. Also, important cryptographic concepts such as computational hardness and cryptographic primitives on which these above-mentioned algorithms are based on have been explained. We gave an explanation of why the security of these algorithms would be breached whenever a sufficiently large quantum computer is available to an adversary. The adversary would use Shor's algorithm, which is powerful because of its polynomial time quantum factorization algorithm, to find the prime factors of a certain composite number.

To counter this breach, we introduced lattice-based cryptography in Chapter 3, a class of post quantum cryptography which consists of algorithms using properties of lattices. We explained the basics of lattice-based cryptography and explained on which hardness it is based on. Also we gave some of its implementations. The well known lattice problems such as the shortest vector problem and the closest vector problem have been addressed and algorithms based on this hardness have been explained.

Lattice-based cryptography holds great promise for the future, arguably the most promising of all post-quantum cryptography classes. Being able to sustain attacks from classical computers and quantum computers, assuming an adversary is computationally limited. There is still much to research on concepts of lattice-based cryptography. Whether it is forming new lattice-based cryptographic primitives or constructing lattice attack-algorithms to break the security of current lattice-based cryptography algorithms, more research in this field could potentially give insights into making lattice-based cryptography even more secure. Another effect is that insights into lattice-based cryptography will cast light on quantum complexity theory. For now, however, lattice-based cryptography remains one of the most promising methods of encrypting information in an upcoming quantum era.

Bibliography

[1] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*. Springer, 2009.

[2] Khan Acadamy Labs, "Time complexity — size of input affects the computation time needed for factorization vs. multiplication," 2017. [Online: `https://www.khanacademy.org/labs/explorations/time-complexity`; accessed June 10th, 2017].

[3] J. van de Pol, "Lattice-based cryptography," Master's thesis, Eindhoven University of Technology — Department of Mathematics and Computer Science, July 2011.

[4] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 6–7, 1982.

[5] N. S. Yanofsky, *An Introduction to quantum computing*. Department of Computer and Information Science, Brooklyn College, CUNY, Brooklyn, N.Y. 11210: arxiv, 2007.

[6] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, pp. 120–126, 1977.

[7] W. Diffie and M. E. Hellman, "New directions in cryptography," *Transactions on Information Theory*, vol. IT-22, pp. 644–654, 11 1976.

[8] D. Augot, M. Finiasz, and N. Sendrier, "A family of fast syndrome based cryptographic hash functions," in *International Conference on Cryptology in Malaysia*, pp. 64–83, Springer, 2005.

[9] D. van Melkebeek, *CS 880: Advanced Complexity Theory: Lecture 13: Average-Case Hardness*. University of Wisconsin Madison, February 2008.

[10] B. Kaliski, "The mathematics of the RSA public-key cryptosystem," *RSA Laboratories*, 2006.

[11] L. S. Hill, "Cryptography in an algebraic alphabet," *The American Mathematical Monthly*, vol. 36, no. 6, pp. 306–312, 1929.

[12] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings, 35th Annual Symposium — Foundations of Computer Science*, pp. 124–134, IEEE, 1994.

[13] J. Kelner, *Topics in Theoretical Computer Science: An Algorithmist's Toolkit - Lecture 19*. Massachusetts Institute of Technology, November 2009.

[14] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pp. 99–108, ACM, 1996.

[15] M. Ajtai, "The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract)," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pp. 10–19, ACM, 1998.

[16] M. Ajtai, R. Kumar, and D. Sivakumar, "A sieve algorithm for the shortest lattice vector problem," in *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pp. 601–610, ACM, 2001.

[17] C. Peikert, *Lattices in cryptography — Lecture notes 2: SVP, Gram-Schmidt, LLL*. University of Michigan, Fall 2013.

[18] O. Goldreich, D. Micciancio, S. Safra, and J. P. Seifert, "Approximating shortest lattice vectors is not harder than approximating closest lattice vectors," *Inf. Process. Lett.*, vol. 71, no. 2, pp. 55–61, 1999.

[19] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems.," in *CHES*, vol. 7428 of *LNCS*, pp. 530–547, Springer, 2012.

[20] H. Lenstra, A. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients.," *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.

[21] L. Babai, "On Lovász lattice reduction and the nearest lattice point problem," *Combinatorica*, vol. 6, no. 1, pp. 1–13, 1986.

[22] P. Nguyen, "Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from CRYPTO97," in *Annual International Cryptology Conference*, pp. 288–304, Springer, 1999.

[23] D. P. Chi, J. W. Choi, J. San Kim, and T. Kim, "Lattice based cryptography for beginners.," *IACR Cryptology ePrint Archive*, vol. 2015, p. 938, 2015.