Dibris

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS)

Homomorphic Signatures over Lattices

by

Stefano Ottolenghi

Master Thesis

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS)

MSc Computer Science
Data Science and Engineering

Homomorphic Signatures over Lattices

by

Stefano Ottolenghi

Advisors: D. Fiore, G. Lagorio Examiner: G. Chiola

September, 2019

Abstract

Digital signatures are used to guarantee the integrity of data. In this work we are concerned with homomorphic signatures over lattice spaces. There has been increasing interest in lattices for cryptographic applications as they seem to be quantum-resistant and efficient in terms of computation. Homomorphic schemes, on their part, are drawing more and more interest as they allow to perform computations on encrypted or signed data, thus being very suitable for cloud applications.

With a homomorphic signature scheme, it is possible to delegate the computation of a given function to an untrusted third party. The third party not only produces the requested result, but attaches to it a signature (that depends both on the data set and the function to be computed). Then, the scheme provides the ability to verify whether the provided result is really correct, or whether the third party cheated.

We first summarize the theory of lattices and its hard problems, and present trapdoor functions that allow to use them for cryptography. Then, we illustrate a homomorphic signature scheme for single bits over the integer-modulo lattices. We finally show how to port this scheme over polynomial rings, with the aim of improving its performance and achieving more functionality. In the end, we discuss some possible further improvements to the scheme by relaxing its guarantee of correctness requirement to a *probabilistic*-correctness.

Contents

Introduction

Imagine to own a huge data set, on which you would like to perform some computation. It could be any operation: either searching through it, or performing some mathematical task like an *average*, for example. Moreover, assume that the computation is very heavy, and resources more powerful than your computer are needed. You thus decide to hand the data over to some cloud provider that will make such a computation for you.

However, you have a very basic requirement: you do not want the third party to cheat on you. That is, you want to be able to verify that the final result is correct. In other words, you want to be sure that whatever result the cloud provider *claims* to be *correct*, it really is. The third party must not purposefully try to deceive you with wrong information, nor to save computing resources by just guessing or approximating the result.

Put in another way, as illustrated in Figure 1, *how can we delegate the computation to a third party, being sure that it does not cheat on the result?*

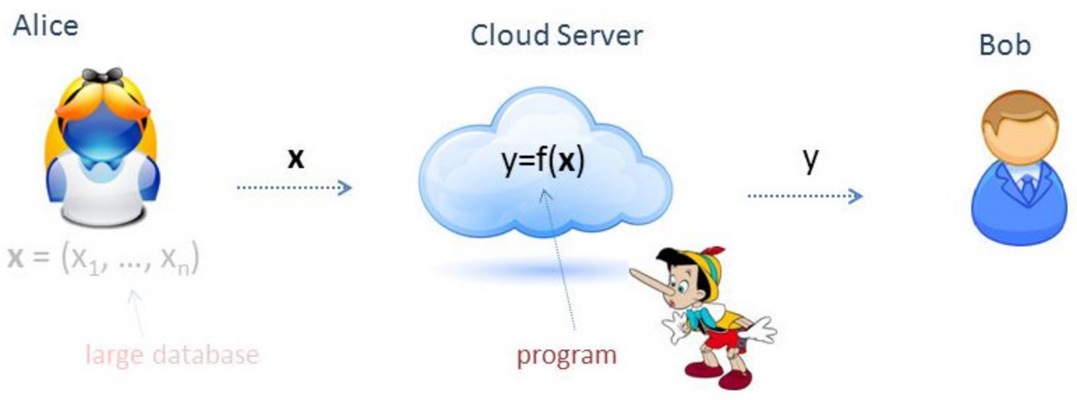

Figure 1: Motivation for homomorphic signatures applied to cloud computation.

Homomorphic signatures allow to achieve this goal. The idea is that every entry in the data set will also carry a corresponding signature, computed by the data owner. The third party then not only produces the *result* of the computation, but attaches to it the result of "corresponding" computations on the original signatures. The party who requested the computation can then verify that the result is correct by checking that the attached signature matches its expectations.

This primitive is called *homomorphic signature* because it allows:

- to verify that something is what it claims to be (*signature*);
- to perform operations on the signed data and manipulate the data signatures accordingly (*homomorphic*).

Notice how the data is never needed for verification: in fact, *everybody* can verify the correctness of the computation given the result, its signature and the signatures for the original data. In other words, to verify the result, we do not need to know the data it originated from - only the relative signatures are needed.

We will not deal with it much in this work, but it is worth mentioning the existance of homomorphic encryption as well (see Gentry's work [Gen09]), as showcased in Figure 2. These schemes allow queries to be made on the ciphertexts. In fact, they allow to manipulate the encrypted data in such a way that, when decrypted, it will yield the expected cleartext data, with the desired operations applied to it.

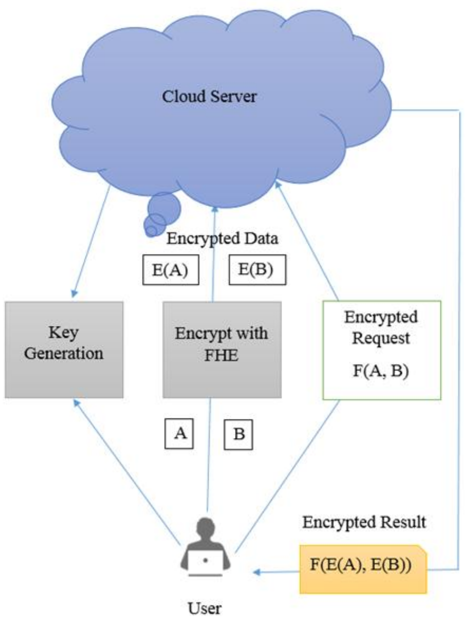

Figure 2: With homomorphic encryption schemes, a user makes a query on the encrypted data stored in the cloud, and obtains a ciphertext (opaque to the server) with the result.

As it happens for traditional schemes, encryption and signatures schemes are complementary, and are both needed to guarantee security, so the use of one does not exclude the other. In fact, while encryption provides confidentiality, signatures are a means of checking data integrity.

These modern schemes have been developed on lattice spaces, and rely on the their hard problems and apt trapdoor functions. Lattice cryptography is becoming more and more popular because of both its simplicity and its supposed strength to quantum computers. In fact, there are no known quantum algorithms that would make lattice hard problems easier, while Shor's algorithm [Sho82] would make RSA (and all schemes that rely on the integer factorization problem) incredibly weak in a quantum world.

Current issues and further research

Homomorphic schemes are quite recent. For this reason, there is still a lot of work to make them efficient and strong enough to be used in real life scenarios and mainstream applications.

Here we briefly list the main issues currently affecting the homomorphic signature scheme [GVW15], which is the main object of this thesis. We will go in more detail in later sections.

1. The number of operations allowed on signed data is limited. This means that it is not possible to perform *any computation* on *any data set*. If a given bound on the noise generated by the operations is exceeded, then the signature is basically invalidated. This is what we refer to as *leveled* homomorphic signatures.

2. The time needed to verify a signature is comparable to the time needed to perform the computation itself. That is, up to now there is no real advantage in delegating the computation versus performing it, other than not having to retain the data set long-term.

3. It is only possible to sign one bit at a time, and the whole scheme works on bits only. This is clearly inconvenient for huge data sets, making it impractical for real-world applications.

4. The signature size is significant. In current schemes, the signature for *one single bit* is a matrix in the space $\mathbb{Z}_q^{n\times m}$; thus significantly disproportionate.

Tackling all these problems are further research topics.

In our work, we tried to improve on points three and four, by moving from the ring $\mathbb{Z}_q$ of integers modulo q to the quotient ring of polynomials $\mathbb{Z}_q[X]/(X^d - 1)$. We also provided some hints to achieve possible improvements on point two, by relaxing the requirement of *correctness* of the result to a *probabilistic correctness*. The discussion on the differences between the schemes on the two rings can be found in Section 3.3.2, while the ideas about probabilistic correctness are detailed in Section 3.2.

As for the structure of this work, in Chapter 1 we present lattice spaces, on which the signature scheme is based, and illustrate hard problems defined on them. In Chapter 2, we define lattice trapdoor functions, that allow exploiting lattice hard problems for cryptographic applications. Finally, in Chapter 3, we present the homomorphic signature scheme [GVW15] and its extension to polynomial rings, discussing the differences between the two instantiations.

Chapter 1

Lattices

The interest in lattices for cryptographic applications has risen in the last few years. This is due to several reasons.

First, as we will shortly explore in more detail, lattices are basically the discrete counterpart of vector spaces. Cryptography has always been based on discrete spaces, and this provides an apt setting. In fact, some problems that may be easy in continuous settings become excruciatingly difficult (if not impossible) in the discrete counterpart.

For example, there are computationally efficient ways to calculate the logarithm in $\mathbb{R}$, but there are no known ways to do the same in discrete groups. It simply does not seem to be possible to port the algorithm from the continuous to the discrete. Integer programming is another instance of a problem that is significantly easier in the continuous setting. In our case, the Gram-Schmidt algorithm [Wik19a] allows to obtain an orthonormal basis from an arbitrary one of $\mathbb{R}^n$, but the same algorithm does not work in $\mathbb{Z}^n$, and there does not seem to be an efficient alternative.

Second, quantum computation does not seem to help in solving hard problems on lattices more efficiently. In other words, lattice cryptography appears to be *quantum-resistant*. In fact, while *there are* quantum algorithms (Shor's algorithm [Sho82]) that would make RSA weaker, for example, there is nothing similar for hard problems over lattices. For this reason, lattice cryptography belongs to the *post-quantum cryptography* field, since its schemes may become very useful in a possible quantum-future. In fact, although they are inherently classic, and do not require anything more than a current PC to run, they may withstand the quantum revolution.

What makes schemes over lattices particularly promising, with respect to other post-quantum schemes, is that other schemes often require *something more* than what our current technology supports. For example, photon-based cryptography [NC10, chapter 12.6] requires a photon polarizer.

In addition, the basic structure of the schemes on lattices is simple and efficient, only involving linear operations that can be highly parallelized to achieve better performance.

Finally, lattices have been shown to support *homomorphic schemes*, both for signatures and for encryption purposes. In fact, the first ever homomorphic encryption scheme was devised in 2008 by Gentry and is based on lattices [Gen09]. This was a major breakthrough and solved what has been regarded as a *holy grail* of cryptography for a long time.

Homomorphic schemes are becoming more and more important as the need for privacy and the use of cloud computation increases, as they allow to perform some computation on the encrypted/signed data. For cipher-schemes, a cloud provider can be instructed to perform some actions on the data *without knowing* what data it is manipulating, but still producing a final (encrypted) result that, when decrypted with the secret key, will yield the correct result! Something similar happens for signature schemes, where the signatures on the initial data set are manipulated to produce a final signature on the result that guarantees that such a result is correct.

Of course, there are downsides. The main disadvantage of current lattice schemes is that, however fertile the field may seem, they are still a long way from having practical implementations. In fact, from both storage and computational perspectives, they are not efficient enough. At any rate, apart from some exceptions like NTRUEncrypt (see Section 1.4), there is still a good amount of work to be done before these schemes become ready for mainstream applications.

1.1 Notation

We will use a bold font for vectors $\mathbf{v}$, while ordinary letters will be used lowercase z for scalars and uppercase A for matrices (with the identity matrix of size n denoted as I_n). B^{-t} corresponds to the transposed of the inverse of B.

Also, we will denote with *poly*(n) a function that is polynomial in n, and with $||U||_\infty$ the norm infinity of a matrix, corresponding to its maximum entry. If not specified otherwise, $||\cdot||$ denotes the norm 2.

For more mathematical background, see the Appendix.

1.2 Basic definitions

Before stating the formal definition of a lattice, let us give a more informal intuition. In its simplest terms, a lattice is a grid of points, as showcased in Figure 1.1. Lattices are the discrete counterparts of vector spaces, with which they share many similarities. Since all lattices are isomorphic to the additive group $\mathbb{Z}^n$, one can think of a lattice as a multi-dimensional cartesian space made of integer-coordinate points only.

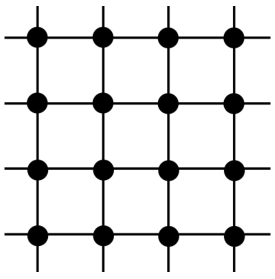
Figure 1.1: A 2D lattice.

Definition 1.2.1 (Lattice). *Given a group $\mathcal{G}$, a lattice $\mathcal{L}$ is a discrete subgroup of $\mathcal{G}$, where discrete means that for any $\mathbf{v} \in \mathcal{L}$, it exists a neighborhood U of $\mathbf{v}$, with $U \subset \mathcal{G}$, such that $U \cap \mathcal{L} = \mathbf{v}$.*

For most of our work, we will consider the case $\mathcal{G} = (\mathbb{R}^n, +)$. In this context, a lattice is a subset of points of $\mathbb{R}^n$ that have some regular spacing between them. Notice that the spacing can be different for each dimension: for example, in two dimensions, the horizontal and vertical spacing sizes can differ.

Another useful way of thinking about a lattice is as a regular tiling of the space $\mathbb{R}^n$. The *tiles* are also called *fundamental domains* of the lattice. Thinking in two dimensions, a lattice consists of a tiling with parallelograms (of any size, but always the same, of course; see Figure 1.2). In higher dimensions, a lattice can be thought of as dividing the space into equal polyedra (i.e. a n-dimensional parallelepiped).

Example 1.2.1. The most intuitive lattice is $\mathbb{Z}^n$, which perfectly fits with the grid representation. Any other lattice (as long as it is a subgroup of $\mathbb{R}^n$) can be obtained from $\mathbb{Z}^n$ through a linear transformation.

Moreover, given any lattice $\mathcal{L}$, another one can easily be built by scaling it of a real factor c, obtaining the lattice $c\mathcal{L}$. This corresponds to uniformly stretching the lattice in all dimensions. □

Lattice basis

Up to now we have given several *interpretations* of a lattice, but they do not tell much about the *process* that can be used to build one. Indeed, one can define a lattice from any basis of $\mathbb{R}^n$, in a similar way as it is done for a vector space.

Definition 1.2.2 (Lattice basis). *Given a set of independent vectors $\mathcal{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\} \subset \mathbb{R}^n$, the lattice generated by them is*

$$\mathcal{L}(\mathcal{B}) = \{\sum_{i=1}^{n} z_i \mathbf{b}_i,\ z_i \in \mathbb{Z}\}$$

Notice how the coefficients of the linear combination are integers, while the basis vectors can have non-integer components: this means that the lattice points can have non-integer coordinates.

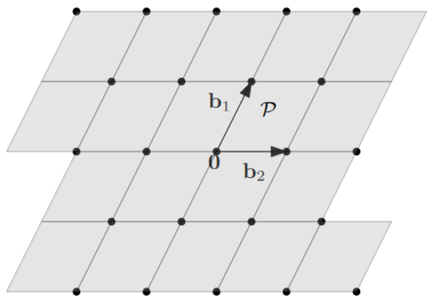

Figure 1.2: An example tiling for a 2D lattice. Black dots are the lattice points, while grey parallelepipeds are lattice fundamental domains.

Example 1.2.2. $\mathcal{L}(1) = \mathbb{Z}$, and $\mathcal{L}((1,0),(0,1)) = \mathbb{Z}^2$.

However, notice how $\mathcal{L}((\frac{1}{2},0),(0,\frac{3}{4}))$ is not $\mathbb{Z}^2$, but only *homomorphic* to it. We can find a bijection between the two, but in the former there are lattice points with non integer coordinates. □

A lattice can be generated by several different bases (the number of which grows exponentially in the lattice dimension). In particular, given a basis $\mathcal{B}$, any unimodular matrix $U \in \mathbb{Z}^{n\times n}$ allows to generate a different basis $\mathcal{B}' = \mathcal{B}U$ for the same lattice. However, there is something that does not change across bases. That is, the absolute value of the determinant of the basis vectors $\mathbf{b}_i$ is constant. In fact, if the basis vectors are arranged in a matrix $\mathbf{B}$, its determinant is invariant with respect to different basis for a given lattice.

Proposition 1.2.1 (Relation between bases). *There exists a unimodular transformation that links two different bases of the same lattice.*

Formally, given an n-dimensional lattice $\mathcal{L}$, if B and B' are both bases of $\mathcal{L}$, then there exists an integer matrix $U \in \mathbb{Z}^{n\times n}$ such that $B = B'U$ and $\det(U) = \pm 1$.

Remark 1.2.1. *Although several bases exist for the same lattice, not all of them are equal applications-wise. In fact, when dealing with cryptographic matters, we make a distinction between "good" and "bad" bases. We defer a more detailed treatment of the differences to Section 1.3.3.*

Remark 1.2.2. $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ *is always a subgroup of* $\mathbb{R}^n$, *but it is not necessarily discrete. For example,* $\mathcal{L}(1, \sqrt{2})$ *is not discrete. It can be shown that a basis generates a discrete subgroup if the generators are all independent with each other [Unk11].*

Finally, it will be useful, later on in our discussion, the definition of the parallelepiped of a given basis. Intuitively, it corresponds to building a parallelepiped centered at the origin, with side lengths equal to half of the basis vectors.

Definition 1.2.3. *The origin-centered parallelepiped of a basis V is defined as:*

$$\mathcal{P}_{1/2}(V) = V \cdot \left[-\frac{1}{2}, \frac{1}{2}\right)^n$$

Minimum distances

Finally, it is useful to define the vector of minimum distance of a lattice. That corresponds to the shortest non-zero vector belonging to the lattice.

Definition 1.2.4 (Minimum distance). *The minimum distance of a lattice $\mathcal{L}$ is*

$$\lambda_1(\mathcal{L}) = \min_{\mathbf{v}\in\mathcal{L}, \mathbf{v}\neq 0} ||\mathbf{v}||$$

It is possible to provide a bound on $\lambda_1(\mathcal{L})$, although not particularly helpful in practice.

Theorem 1.2.1 (Minkowski's First Theorem). *For any full-rank lattice $\mathcal{L}$ of rank n,*

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}(\det(\mathcal{L}))^{\frac{1}{n}}$$

Notice how the vector having length $\lambda_1(\mathcal{L})$ is (always) not unique. In fact, since $\mathcal{L}$ is a subgroup, if $\mathbf{v}$ is a vector of minimum distance, then $-\mathbf{v}$ is as well, and belongs to $\mathcal{L}$. In other words, there are always at least two elements having length $\lambda_1(\mathcal{L})$. However, since for practical applications the exact element of length $\lambda_1(\mathcal{L})$ that is used is rarely important, we often refer to the vector of minimum distance $\lambda_1(\mathcal{L})$ as one (*any*) of the possible ones.

It is also possible to define the i-th successive minimum $\lambda_i(\mathcal{L})$. Intuitively, it captures the idea of *how far we need to look* in order to obtain a space of dimension i. More formally,

Definition 1.2.5 (i-th successive minimum). $\lambda_i(\mathcal{L}) = \min r$ *such that* $\mathcal{L}$ *has* i *linearly independent vectors of norm at most* r*. In other words,* $\lambda_i(\mathcal{L})$ *is the radius of the smallest sphere containing* i *linearly independent non-null lattice points.*

A comprehensive survey on lattices and their foundations for cryptographic schemes can be found in [Pei16].

1.3 Hard problems

Lattices are interesting for cryptography applications because of the *hard problems* that can be defined on them. These are problems that are *believed* to be computationally hard to solve, and can be used to build encryption and signature schemes.

1.3.1 The Shortest Vector Problem

The fundamental hard problem is finding a vector of minimum distance. Together with its variants, it is the foundation for the more modern hard problem *Short Integer Solution* described in Section 1.5.1.

Definition 1.3.1 (Shortest Vector Problem (SVP)). *Given an arbitrary basis* $\mathcal{B}$ *of* $\mathcal{L}$*, find an element* $\mathbf{v} \in \mathcal{L}$ *such that* $\mathbf{v} = \lambda_1(\mathcal{L})$*.*

We stress the fact that the problem relies on an *arbitrary* basis. In fact, there are bases for which solving the problem may be easy, as long as one knows that the basis they have is optimal. In general, however, this is a hard problem.

Intuitively, solving the *Shortest Vector Problem* for a given lattice corresponds to finding the minimum distance between any two lattice vectors, as illustrated in Figure 1.3. Thinking of a lattice as a grid, we are looking for the smallest step size.

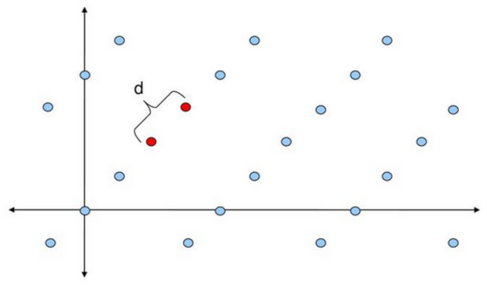

Figure 1.3: An example instance of the *Shortest Vector Problem*, with the solution having length d. [LM18]

Remark 1.3.1. *One may wonder whether the problem is well-posed at all, i.e. whether it always admits a solution. The answer is yes. We skip the proof, which can be found in [Unk11].*

Example 1.3.1. Suppose to have the lattice generated by $\mathcal{B} = \{(1,0),(0,1)\}$, which is basically equal to $\mathbb{Z}^2$. Imagine to be looking for its minimum distance. If we are the authors of the problem, and we have the basis $\mathcal{B}$, then we immediately know that there cannot be any vector with $||\cdot|| \leq 1$, with the exception of the zero vector. So with this basis, solving the *Shortest Vector Problem* is straightforward: both basis vectors are already solutions!

However, several bases exist for this lattice. For example, another is $\mathcal{B}' = \{(9,4),(11,5)\}$, which makes a unimodular matrix. With this basis there does not seem to be any quick way to find a Shortest Vector. If we knew that the solution is $(1,0)$, then we could simply solve the linear system

$$\begin{cases} 1 = 9j + 11l \\ 0 = 4j + 5l \end{cases}$$

and discover the exact combination of the basis vectors that yield the solution (in this case, $l = -4$ and $j = 5$).

But suppose we were challenged to solve this problem given the $\mathcal{B}'$ basis as only information (i.e. no solution and no basis $\mathcal{B}$). What we would have to find is some combination of the basis vectors that yields shortest size. That is, finding apt y, z (with the point (y,z) belonging to the lattice) such that, for some $j, l \in \mathbb{Z}$, they are solution of the following linear system:

$$\begin{cases} y = 9j + 11l \\ z = 4j + 5l \end{cases}$$

where having shortest size means that $||(y,z)||$ is the smallest (nonzero) we can find in the whole lattice.

This is an under-determined system of equations: it has four unknowns, but only two equations. Usually, these kind of systems have infinite solutions. However, constraints on the solution have an impact in this case, and the solution space is actually finite (although it may be big if the lattice dimension is such).

Finally, notice how j and l do not have any bounds: they can be arbitrarily big. We will come back to this remark later, when talking about the *Bounded Distance Decoding* problem in Section 1.3.3; anyway, at least intuitively, we already see that from a cryptographic prospective, *not all bases are equal.* In fact, some of them may allow for fast solution of some problem, while some others may make it almost impossible. $\square$

1.3.2 Approximate problems

The *Shortest Vector Problem* is the ancestor of all other hard problems. All other hard problems, in a way or another, require solution of this one. That is, other hard problems exist whose hardness is based on the SVP.

Definition 1.3.2 (Approximate Shortest Vector Problem (SVP$_\gamma$)). *Given an arbitrary basis $\mathcal{B}$ of an n-dimensional lattice $\mathcal{L}$, find a $\mathbf{v} \in \mathcal{L}$ such that $\mathbf{v} \leq \gamma\ \lambda_1(\mathcal{L})$.*

The approximate problem is particularly useful in cryptographic applications, in which the parameter γ is usually a function of the lattice dimension n. This ensures that γ is not fixed and changes according to the security parameter of the setting.

There is also a decisional variant of the *Approximate Shortest Vector Problem*, in which we are asked to provide a guess about the size of the shortest vector of $\mathcal{L}$.

Definition 1.3.3 (Decisional Approximate Shortest Vector Problem (Gap-SVP$_\gamma$)). *Given an arbitrary basis $\mathcal{B}$ of an n-dimensional lattice $\mathcal{L}$ and knowing that $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma(n)$, decide which is the case.*

The decisional version is particularly important because several cryptosystems rely on it for security, while a proof based on its search counterpart has not been devised yet.

1.3.3 The Bounded Distance Decoding Problem

The *Bounded Distance Decoding* problem is the last fundamental hard problem on lattices that we cover, and possibly the most elaborate. It is also the most recurrent when it comes to cryptographic applications, and the basis for the more modern hard problem *Learning With Errors* described in Section 1.5.2.

In this problem, we are given a lattice basis and a challenge point $\mathbf{x}$ in space (that does not necessarily belong to the lattice). The problem is to determine the lattice point to which $\mathbf{x}$ is closest to. An example instance in shown in Figure 1.4.

Definition 1.3.4 (Bounded Distance Decoding Problem ($\mathcal{BDD}(\mathcal{L}, \mathbf{x})$)). *Given an arbitrary basis $\mathcal{B}$ of an n-dimensional lattice $\mathcal{L}$ and a challenge point $\mathbf{x} \in \mathbb{R}^n$, with the guarantee that it exists a unique vector $\mathbf{v} \in \mathcal{L}$ s.t. $||\mathbf{x}, \mathbf{v}|| \leq d = \frac{\lambda_1(\mathcal{L})}{2}$, find $\mathbf{v}$.*

Notice that bounding d as above is necessary to guarantee the uniqueness of the solution. If one imagines to be standing on $\mathbf{v}$ (the problem solution), the target points that can be uniquely linked to it are within $\frac{\lambda_1(\mathcal{L})}{2}$ distance. Anything farther than that *could be* closer to a different lattice point.

Another way of seeing the problem is that of a *decoding problem*, as the name suggests. With this interpretation, we imagine to have a lattice point $\mathbf{v}$ and perturb it with some small noise $\mathbf{e}$. Given the sum $\mathbf{v} + \mathbf{e} \notin \mathcal{L}$, we are required to *read through the noise* and recover the original lattice point. As we will shortly see, different bases can have a different impact on the ease of solution.

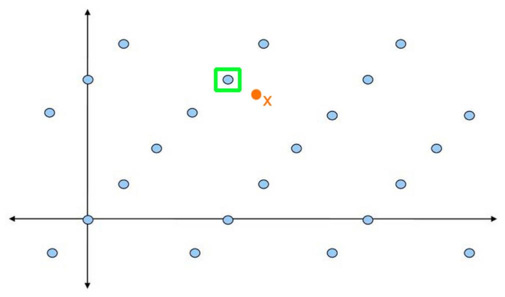

Figure 1.4: Example instance of the *Bounded Distance Decoding Problem.* The red point is the challenge point, the solution is marked in green. [LM18]

1.3.3.1 What makes the Bounded Distance Decoding Problem difficult?

As the literature goes, the *Bounded Distance Decoding Problem* is hard in general, but easy if the basis is *nice enough.* In the case of encryption, for example, the idea is that we can encode the secret message as a lattice point, and then add to it some small noise (i.e. a small element $\mathbf{e} \in \mathbb{R}^n$). This generates an instance of the BDD problem, and then the decoding can only be done by someone who holds the good basis for the lattice. Instead, the parties who have a bad basis are going to have a hard time decrypting the ciphertext.

Since the *Bounded Distance Decoding Problem* is vital to several homomorphic encryption and signatures schemes, it is worth spending some time to fully understand it. Of course, no proof of hardness can be given, as this is a problem that is only *believed* to be hard, but we would like to at least get an *idea* of what makes it hard.

Solving the BDD Problem using a good basis

It is very tricky to define what a "good basis" is. The reason is that, generally, *good* is only defined with respect to *bad*. In definition 1.3.6 we will get to our final definition of what a "good basis" is, but that is only the landmark of our route. We will get there eventually, but first let us give a poorer (although not necessarily easier) definition of what a "good" lattice basis is.

Definition 1.3.5 (Good basis, temporary). *A basis $\mathcal{B}$ for a lattice $\mathcal{L}$ is good if it is made of orthogonal and short vectors.*

What *short* vectors means is pretty vague. Usually, in cryptographic applications, a lattice is *generated* from a (good) basis. Its vectors are considered to be short if they are significantly shorter than vectors of any other basis.

Before considering a general example, let us consider a more specific (although tighter) case. Consider a base in which each of its $\mathbf{b}_i$ is of the form $(0, ..., 0, k_i, 0, ..., 0)$, where $k_i \in \mathbb{R}$ is in position i. In other words, we are considering the canonical basis of $\mathbb{R}^n$, in which each vector has been re-scaled by an independent real factor.

Example 1.3.2. Consider $\mathbb{R}^2$, with the lattice $\mathcal{L}$ generated by $\mathbf{b}_0 = (\frac{1}{2}, 0), \mathbf{b}_1 = (0, \frac{5}{4})$ as basis vectors.

Suppose to be given $\mathbf{x} = (\frac{3}{7}, \frac{9}{10})$ as the *Bounded Distance Decoding* challenge point. It does not belong to $\mathcal{L}$, but it is only $(\frac{1}{14}, \frac{9}{25})$ away from the point $(\frac{1}{2}, \frac{5}{4})$, which *does* belong to the lattice. Figure 1.5 shows the settings.

Let us now formalize the problem to be solved: we are looking for the lattice point closest to $\mathbf{x}$. So, sitting on $\mathbf{x}$, we are looking for the linear combination with integer coefficients of the basis vectors that is closest to us. Breaking it component-wise, we are looking for $\min y, z \in \mathbb{R}$ and generic $k, j \in \mathbb{Z}$ such that they are solution of the following linear system:

$$\begin{cases} \frac{3}{7} + y = \frac{1}{2}k \\ \frac{9}{10} + z = \frac{5}{4}j \end{cases}$$

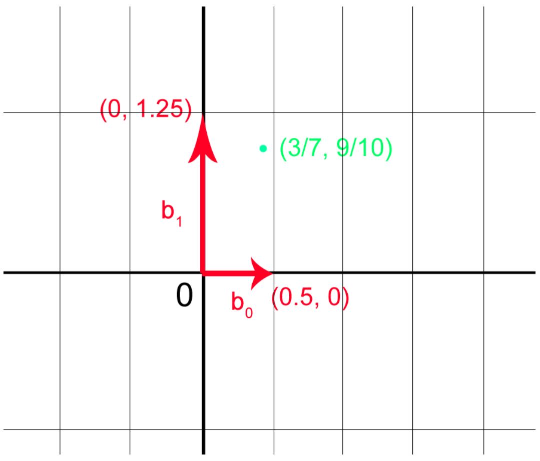

Figure 1.5: The *Bounded Distance Decoding Problem* instance with a good basis (in red). The challenge point is in green. Grid represents lattice points.

This may look like a difficult optimization problem, but it is not! In fact, each of these equations is independent, so they can be solved one by one. The individual minimum problems are easy and can be solved quickly, as they are convex. Also, notice how useful and important it is the guarantee that the solution is close to $\mathbf{x}$: this ensures that k, j will probably be quite small integers, making the algorithm converge very quickly.

As a side remark, notice that it would also be possible to put boundaries on y, z with respect to the size of the basis vectors. However, all that we care about is that *there is* a polynomial algorithm that allows to get to the solution.

Thus, what we obtain is that the overall complexity of solving a *Bounded Distance Decoding Problem* instance *with a good basis* is $\Theta(\Theta(\min) \cdot n)$. $\square$

Solving the BDD problem using a bad basis

A “bad”’ basis is any basis that does not satisfy any of the two conditions of a good basis: it may be poorly orthogonal, or may be made of long vectors. We will later try to understand what roles these differences play in solving the problem: for now, let us just consider an example again.

Example 1.3.3. Another basis for the lattice generated by the good basis in Example 1.3.2 is $\mathbf{b}_0 = (\frac{9}{2}, \frac{5}{4}), \mathbf{b}_1 = (5, \frac{10}{4})$. This is a bad one: vectors are non-orthogonal and quite longer than the nice ones. Refer to Figure 1.6 for a graphical illustration of the setup.

Let us write down the system of equations coordinate-wise as we did for the good basis.

We are looking for min $y, z \in \mathbb{R}$ and $k, j \in \mathbb{Z}$ such that they are solution of:

$$\begin{cases} \frac{3}{7} + y = \frac{9}{2}k + 5j \\ \frac{9}{10} + z = \frac{5}{4}k + \frac{10}{4}j \end{cases}$$

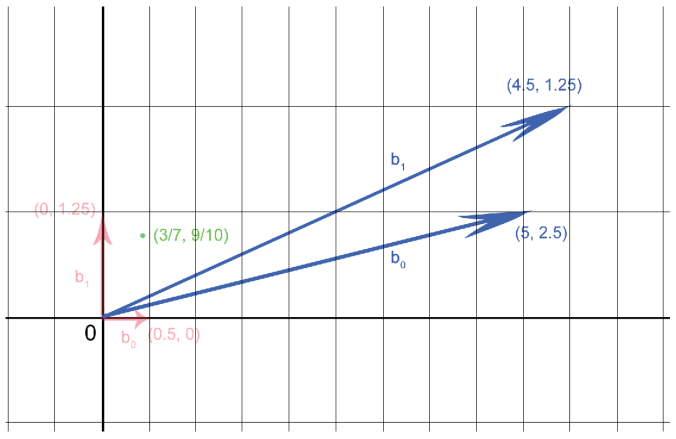

Figure 1.6: The *Bounded Distance Decoding Problem* instance with a bad basis (in blue, shadow of good basis in red). Challenge point in green. Grid represents lattice points.

This may look similar as the previous example, but this time it really is a system, meaning that the equations are no longer independent. In fact, k and j show up in both equations. Thus, the equations cannot be solved separately.

Moreover, we have 4 unknowns and 2 equations. This means that the system is under-determined: in principle, there are infinite solutions. However, we cannot just pick one of the infinite solutions at random: we are trying to find a solution that is constrained to be *minimum*.

Especially with big n, solving this optimization problem can definitely be non-trivial! $\square$

On the differences between good and bad bases

Previous examples (1.3.2, 1.3.3) have shed some light on what distinguishes a good basis from a bad one, when it comes to solving the *Bounded Distance Decoding Problem.* However, we have not really understood how the basis properties influence ease or hardness of solution. This is what this final Section is going to tackle.

In Example 1.3.2 we required the lattice basis to satisfy the condition of being a "stretched" version of the canonical one - i.e. that each vector had only one non-zero entry. This requirement is at the root of the independence of the minimum problems, which in turn allows for easy resolution of the *Bounded Distance Decoding Problem.* Of course, this condition is stronger than simple orthogonality, but we will shortly explain how orthogonality is there to achieve this very same goal as well.

Remark 1.3.2. *Notice how the behavior above would present itself even with the canonical basis, without any re-scaling of the vectors. In other words, using the canonical basis is enough to obtain a set of independent minimum problems that can be solved efficiently.*

However, it is known that when dealing with cryptography matters key re-usage or determinism is insecure. When cryptography is made over lattices, this means we cannot always use the same basis: some randomness is required. We may re-use the same lattice, but not the same basis. *Using a set of independent vectors, each having only one non-zero coordinate achieves both: makes the problem easy* and *avoids re-usage of the key.*

The second requirement we asked for is vector shortness. This does not give an immediate advantage to the party holding the good basis, and is a subtler point to understand. Its role is to make it harder to solve the problem for those owning a bad basis. Very roughly, the idea is that, given a challenge point $\mathbf{x} \in \mathbb{R}^n$, with the short basis vectors we can take small steps from it and look around for the solution among nearby points. Although it may take some time to find the best one, we are guaranteed that, if we take one step at a time in each direction, we will soon find the solution (which, remember, is guaranteed to be close). Instead, if we have long vectors, every time we use one we have to make a *big leap* in one direction.

In other words, the advantage of having the good basis lies in knowing the *step size* of the lattice. This allow to take steps of considerate size, with the guarantee that, if we do not skip any steps, we will stumble upon the solution of $\mathcal{BDD}(\mathcal{L}, \mathbf{x})$. On the other hand, the parties holding the bad basis can only take huge jumps and may have a hard time pinpointing the right point.

Earlier, we mentioned that the role of the re-scaling is akin to that of the orthogonality. Let us now explain why. Let us consider a basis of that kind, such as $\mathcal{B} = \{\mathbf{b}_1 = (\frac{\sqrt{3}}{2}, \frac{1}{2}), \mathbf{b}_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})\}$. This is just the canonical basis tilted 45 degrees counter-clockwise. If we wrote down the minimum problem to solve given a challenge point, we would expect it to be pretty similar to the one with the bad basis we saw in Example 1.3.3, with the equations not being independent; thus, difficult to solve.

In particular, for $\mathbf{x} = (\frac{3}{7}, \frac{9}{10})$, we would be looking for $\min y, z \in \mathbb{R}$ and $k, j \in \mathbb{Z}$ such that they are solution of:

$$\begin{cases} \frac{3}{7} + y = \frac{\sqrt{3}}{2}k + \frac{1}{2}j \\ \frac{9}{10} + z = \frac{1}{2}k + \frac{\sqrt{3}}{2}j \end{cases}$$

However, since if we know that $\mathcal{B}$ is orthogonal, we can transform it into a re-scaled version of the canonical basis. It is beyond the scope of this work to explain how it can be done from a practical point of view; it suffices to say that it is possible to find a rotation matrix U that allows to obtain $\mathcal{B}' = \{\mathbf{v}_1', \mathbf{v}_2'\}$, which generates a new lattice $\mathcal{L}'$. U also allows to obtain a rotated version $\mathbf{x}'$ of the challenge point as well.

Then, it is possible to solve the problem $\mathcal{BDD}(\mathcal{L}', \mathbf{x}')$, obtaining a solution $\mathbf{v}'$. Notice that $\mathbf{v}'$ is not the solution to $\mathcal{BDD}(\mathcal{L}, \mathbf{x})$, but we can easily rotate it back to find the solution $\mathbf{v}$ to the original problem!

Remark 1.3.3. *One may be tempted to think that* $\mathbf{v}'$ *would be the solution to* $\mathcal{BDD}(\mathcal{L}, \mathbf{x})$*, but that is not the case. The reason is that span*$(\mathcal{B}') \neq$ *span*$(\mathcal{B})$*, and thus the two lattices differ.*

As a final note, let us stop to consider whether using a general orthogonal basis over a more specific re-scaled canonical basis has any advantage. It seems that this is not the case, because using a basis of the former kind does not make the opponent's job any harder, but only increases the computational cost for the honest party. For this reason, we argue that there is no good reason to use a random orthogonal basis instead of a stretched canonical one.

Also, notice how having a generic orthogonal basis is not enough for an adversary to break the problem. In fact, if it is orthogonal, but its vectors are long, the problem is still somewhat hard because of the previous observations on vector shortness.

So finally, we are ready to provide a better definition of a good basis, which is what we will rely on from here onwards.

Definition 1.3.6 (Good basis, final). *A good basis for a lattice* $\mathcal{L}$ *is any orthogonal set of vectors that generates* $\mathcal{L}$*, as long as all other ("bad") bases are made of longer and poorly orthogonal vectors.*

Let us stop to notice that, since we require *all other bases* to be worse than the good one, a good basis is essentially unique! That is, *there is only one good basis per lattice* (without accounting for flips), and its vectors encode the lattice step size in each dimension. In other words, the good basis is the shortest one possible. This is unlike vector spaces, in which the continuous space would make it impossible to define a *shortest* basis.

Good and bad bases in cryptography

Up to now, it may not be clear how these hard problems and lattice bases we have discussed so far could be used for cryptographic applications. We defer an in-depth discussion of the matter to Section 2, but we would like to give at least an idea here.

In public-key cryptography, schemes rely on two different keys: one is public and shared with the world, and the other is kept secret. The public key is used to encrypt data, while the secret one allows to decrypt. Similarly, in a signature setting, the public key is needed to verify a signature, while the act of signing can only be performed with the secret key.

These public-key schemes rely on some *trapdoor function*: a function that is easy to compute in one direction (i.e. data encryption/signature verification) but difficult to invert *unless* one has some special information. The special information consists in the *trapdoor*, and is represented by the secret key.

For schemes that operate on lattices, something similar happens. In fact, we have seen that it may be easy to create an instance of some hard problem (using *any* basis of the lattice), for example of the *Bounded Distance Decoding Problem*, but that the only efficient way of solving it is by having a good basis of the lattice. Thus, one can think of the good basis as being a *trapdoor* (i.e. a secret key), while one of the the bad basis as a public key.

1.3.4 Obtaining a good basis from a random one

One immediate question springs to mind: is it easy to obtain a "good" basis starting from an arbitrary, possibly "bad", one? Clearly, the answer *must* be no, otherwise all the problems we have defined as *hard* up to now would become easy. In fact, if that were the case, one could start from any basis, obtain a "good" one, and then solve the problem instance at hand.

For euclidean spaces, the Gram-Schmidt orthogonalization process [Wik19a] allows to build an orthogonal basis of the space starting from a random one. Optionally, the obtained basis can also be normalized. Is it not possible to devise a very similar algorithm for lattice bases? The answer is no.

The point is that to obtain a good basis one must essentially solve a *Shortest Vector Problem.* In fact, the two problems are tightly linked. As we have already remarked, a good basis is unique, and it contains the *step size* in each of the possible lattice "directions". However, obtaining the step size *is* indeed a *Shortest Vector Problem.*

There is an algorithm that allows to build a (quasi) good basis for lattices, which uses the same ideas of the Gram-Schmidt process, but it only runs in (sub-)exponential time. This is the *Lenstra–Lenstra–Lovász algorithm* [Wik19b].

1.4 An example scheme: NTRUEncrypt

Here we present an example scheme on lattices, the *NTRUEncrypt* public-key cryptosystem [HPS98]. It is *relatively* new, as it was first developed (and patented) around 1996. However, it has been constantly improved throughout the last decades, and it is now fully accepted to IEEE P1363 and X9.98 standards.

It is possibly the only scheme on lattices that is currently used in real world scenarios.

In fact, given its speed and low memory usage, it is ideal for mobile devices and smart cards. In fact, the scheme works on polynomials

$$a = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1}$$

in the quotient ring $\mathbb{Z}[X]/(X^N - 1)$ and only involves simple polynomial multiplications, which is significantly faster [YB17] than other asymmetric encryption schemes such as RSA, ElGamal or elliptic curves.

NTRU is parametrized by three integer parameters (N, p, q): the first representing the maximum degree $N - 1$ of the polynomials, the second being a small modulus and the third a large modulus. The constraints are that N is prime, p is smaller than q, and p and q are co-prime. The scheme seems now secure to all known sorts of attacks and efficient enough to be deployed mainstream.

1.4.1 Key generation

Being an asymmetric scheme, both a private and a public key are generated.

We start with two polynomials $\mathbf{f}, \mathbf{g}$ with degree at most $N - 1$ and with coefficients in $\{-1, 0, 1\}$. Additionally, $\mathbf{f}$ must also be invertible modulo p and modulo q, meaning that there must exist polynomials $\mathbf{f}_p, \mathbf{f}_q$ such that $\mathbf{f} \cdot \mathbf{f}_p = 1 \mod p$ and $\mathbf{f} \cdot \mathbf{f}_q = 1 \mod q$. In this setting, the $\mod$ operation acts on the coefficients of the polynomials.

Then, all the polynomials $\mathbf{f}, \mathbf{f}_p, \mathbf{f}_q, \mathbf{g}$ are kept secret and make up the secret key. The public key $\mathbf{h}$, on the other hand, is generated by computing

$$\mathbf{h} = p\, \mathbf{f}_q \cdot \mathbf{g} \mod q$$

1.4.2 Encryption

To send a message, it must first be encoded as a polynomial $\mathbf{m}$ with coefficients in $\{-1, 0, 1\}$. This corresponds to finding the binary or ternary representation of the message. Then, a random polynomial $\mathbf{r}$ with coefficients in $\{-1, 0, 1\}$ is chosen with the role of obscuring the message.

Using the public key $\mathbf{h}$ and the noise $\mathbf{r}$, the ciphertext is computed as such:

$$\mathbf{e} = \mathbf{r} \cdot \mathbf{h} + \mathbf{m} \mod q$$

which is now safe to be sent on a public channel. Of course, care must be taken in keeping the error factor $\mathbf{r}$ secret as well. Otherwise, an adversary could easily compute the product $\mathbf{r} \cdot \mathbf{h}$ and recover the cleartext secret message.

1.4.3 Decryption

To decrypt, the encrypted message $\mathbf{e}$ is first multiplied by the secret polynomial $\mathbf{f}$:

$$\begin{aligned}\mathbf{a} &= \mathbf{f} \cdot \mathbf{e} \mod q \\ &= \mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \mod q \\ &= \mathbf{f} \cdot (\mathbf{r} \cdot p\, \mathbf{f}_q \cdot \mathbf{g} + \mathbf{m}) \mod q \\ &= p\, \mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \mathbf{m} \mod q\end{aligned}$$

because $\mathbf{f} \cdot \mathbf{f}_q = 1 \mod q$.

Next, we reduce the polynomial $\mathbf{a}$ modulo p, exploiting the fact that $p\, \mathbf{r} \cdot \mathbf{g} = 0 \mod p$:

$$\mathbf{b} = \mathbf{a} \mod p = \mathbf{f} \cdot \mathbf{m} \mod p$$

At this point, the original message $\mathbf{m}$ can be recovered using the secret polynomial $\mathbf{f}_p$ and the fact that it is the inverse of $\mathbf{f}$ modulo p:

$$\mathbf{c} = \mathbf{f}_p \cdot \mathbf{b} = \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} \mod p = \mathbf{m} \mod p$$

1.5 Modern hard problems

Modern lattice cryptography is based on hard problems called *Short Integer Solution* (SIS) and *Learning With Errors* (LWE). These are much more mathematically-versed than the “traditional” hard problems we have covered in previous sections, but still rely on their hardness.

1.5.1 The Short Integer Solution Problem

The *Short Integer Solution* problem serves as foundation for most modern lattice-based digital signature schemes, which is the main topic of this thesis. However, it has also been used to build collision-resistant hash functions and identification schemes. It has not been used for encryption, though, for which the *Learning With Errors* problem (see 1.5.2) is more versed.

Informally, the problem asks, given many random elements of $\mathbb{Z}_q$, to find a “short” non-trivial integer combination of them that sums to zero. More formally,

Definition 1.5.1 (Short Integer Solution ($\mathbf{SIS}_{n,q,\beta,m}$)). *Given a matrix $A \in \mathbb{Z}_q^{n\times m}$ (or, which is the same, m random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$), find a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^m$ satisfying $||z|| \leq \beta$ such that*

$$A\mathbf{z} = \sum_i^n \mathbf{a}_i \cdot z_i = \mathbf{0} \in \mathbb{Z}_q^n$$

Remark 1.5.1. *Notice how the problem becomes easy if the constraint on the norm $||\mathbf{z}|| \leq \beta$ is removed. In fact, the equation $A\mathbf{z} = 0$ alone can be solved efficiently via Gaussian elimination for linear systems.*

Remark 1.5.2. *Depending on the tightness of the bound β, the solution may not be unique. For example, if $\mathbf{z}$ is a solution, with $||\mathbf{z}|| \leq \frac{\beta}{2}$, then $2 \cdot \mathbf{z}$ is a solution as well.*

1.5.1.1 On the choice of parameters

The problem is defined with respect to four different parameters: n, q, β, m. In the following remarks we shall briefly discuss their roles and relationships.

Remark 1.5.3 (On parameters n and m). *If we can solve the problem for a matrix A, then we can do so for any extension $[A \mid A']$. In fact, it is enough to pad the solution vector with zeros to obtain a solution for the extended matrix, as this does not change its norm.*

The consequence is that we may ignore columns $\mathbf{a}_i$ as wished. This means that a SIS instance can only get easier as m increases, while gets more difficult as n increases.

Proposition 1.5.1 (On parameter β and solution existance). *If $\beta \geq \sqrt{\bar{m}}$ and $m \geq \bar{m}$, with $\bar{m}$ the smallest integer greater than $n \log q$ (i.e. $\lceil n \log q \rceil$ if not integer, itself incremented by one otherwise), then the instance $SIS_{n,q,\beta,m}$ is guaranteed to admit a solution.*

Proof. Suppose $m = \bar{m}$ (this does not affect generality, as the argument works for any greater m because of the previous remark).

Now notice that there are more than q^n vectors of the form $\{0,1\}^m$, because

$$|\{0,1\}^m| > |\{0,1\}^{n \log q}| = q^n.$$

Then there must be two distinct $\mathbf{x}, \mathbf{x}' \in \mathbb{Z}^m$ such that $A\mathbf{x} = A\mathbf{x}'$. Their difference $\mathbf{x} - \mathbf{x}'$ belongs to $\{0, \pm 1\}^m$ and is thus a solution of norm at most β. □

Remark 1.5.4. *It is interesting to notice how the argument of the proof of proposition 1.5.1 suggests that the SIS problem may support the construction of collision-resistant hash functions, if we assume its hardness. In fact, considering the function family $\{f_A : \{0,1\}^m \to \mathbb{Z}_q^n\}$, a collision for f_A would immediately yield a solution for the SIS problem on A. Moreover, given the hardness of the problem, these functions would also be one-way, thus satisfying the definition of hash functions.*

It is beyond the scope of this work to investigate how this construction could be done: more information can be found in [PR06].

Finally, it is worth spending some words on the hardness of the problem. In fact, for what choice of parameters is the *Short Integer Solution* problem hard? The main parameter that controls the hardness of the problem is n, but q also plays a role as stated in the next result.

Proposition 1.5.2. *If $q \geq \beta n^\delta$, for any integer $\delta > 0$, then the corresponding SIS instance is hard. On the other hand, for $q < \beta$ it becomes trivial [MP13].*

Proof. On the ease of solution if $q < \beta$, notice that, if β is allowed to be q, then

$$\mathbf{z} = (q, 0, \cdots, 0)$$

is a valid (and trivial) solution.

For the proof of hardness with $q \geq \beta n^\delta$, see [MP13]. □

Example 1.5.1. Let us consider $\mathbb{Z}_6$, with samples $\mathbf{a}_0 = (3), \mathbf{a}_1 = (2), \mathbf{a}_2 = (5)$. Of course, this is by no means a cryptographic setting, but rather a toy example to better understand the problem.

By proposition 1.5.1 and with our current parameters, we have

$$\bar{m} = n \log q = 1 \log 6 \approx 2.5.$$

Let us go through the conditions that guarantee the existence of a solution:

- $m \geq \bar{m}$: we provided three samples $\mathbf{a}_0, \mathbf{a}_1$ and $\mathbf{a}_2$, so this is trivially satisfied;
- $\beta \geq \sqrt{\bar{m}}$: since $1 < \sqrt{\bar{m}} < 2$, we need to set $\beta \geq 2$.
  Let us set it to its tightest value then: $\beta = 2$.

The (only) solution to this instance of the SIS problem is

$$\mathbf{z} = (2, 0, 0),$$

for which $2 = ||\mathbf{z}|| \leq \beta = 2$ and $A\mathbf{z} = (3, 2, 5)^T(2, 0, 0) = 6 = 0 \mod 6$. □

1.5.1.2 SIS on lattices

What is most important is that the *Short Integer Solution* problem can be seen as a *Short Vector Problem* on a certain family of lattices. These are called q-ary m-dimensional lattices, and they are defined as such:

$$\mathcal{L}^{\perp}(A) = \{\mathbf{z} \in \mathbb{Z}^m : A\mathbf{z} = \mathbf{0} \mod q\}$$

Notice how the space $\mathcal{L}^{\perp}(A)$ is basically the *solution space* of SIS on the matrix A. They are called q-ary because $\mathcal{L}^{\perp}(A)$ contains a copy of $q\mathbb{Z}^m$ (since any element in $q\mathbb{Z}^m$ is congruent to 0 modulo q), and are m-dimensional because they contain the vectors $q \cdot \mathbf{e}_i$ for each $i \in [1, \cdots, m]$ (since $q \cdot \mathbf{e}_i = 0 \mod q$), where $\{\mathbf{e}_i\}$ is the standard basis of $\mathbb{Z}^m$.

The link between the *Short Integer Solution* problem and the *Minimum distance* one allows to reduce the former to the more classical hard problems, as stated in the next theorem. This is important as it means we only have to rely on the hardness of the *Shortest Vector Problem*, while still using the more mathematical-versed and easily implementable SIS problem.

Theorem 1.5.1. *For any $m, \beta = poly(n)$ and prime $q \geq \beta \cdot \omega(\sqrt{n \log n})$, the average-case problem $SIS_{n,q,\beta,m}$ is as hard as approximating the Shortest Independent Vectors and Decisional Shortest Vector problems.*

1.5.1.3 Normal form

It is possible to achieve a (possibly) significant optimization in terms of size of the matrix A at no cost in cryptographic hardness. The only assumption we need to make is that it must be possible to rewrite A as $A = [A_1 \mid A_2]$, with $A_1 \in \mathbb{Z}^{n \times n}$ being invertible. This does not affect generality, as from remark 1.5.3 we know that columns can be ignored (also, re-ordering the columns does not affect the norm of the solution).

Then, we replace A with

$$A' = A_1^{-1} \cdot A = [I_n \mid \bar{A} = A_1^{-1} A_2]$$

The matrix I_n is then implicit, and does not need to be distributed. In fact, the instance of the SIS problem is fully embedded into $\bar{A}$, which is of size $(m-n) \times n$. What is most important is that this transformation does not affect the SIS solutions, as we shall shortly prove. Moreover, notice that $\bar{A}$ is uniformly random, as A_2 is uniform and independent of A_1^{-1}.

Proposition 1.5.3. *A and A' have exactly the same set of SIS solutions.*

Proof. Taking advantage of the fact that A_1 is invertible,

$$A\mathbf{z} = \mathbf{0} \mod q \Leftrightarrow A_1^{-1} \cdot A\mathbf{z} = \mathbf{0} \mod q \Leftrightarrow A'\mathbf{z} = \mathbf{0} \mod q$$

□

1.5.1.4 Inhomogeneous SIS problem

The formulation of the *Short Integer Solution* problem we have given so far is also called homogeneous, being in the form $\sum_i^n \mathbf{a}_i \cdot z_i = \mathbf{0}$. However, it can also be stated in the *inhomogeneous form*, which is more generic. In this case, we are not looking for an integer combination of the $\mathbf{a}_i$ that sums to zero. Instead, the examples are required to sum to some given non-null element of $\mathbb{Z}_q^n$.

Definition 1.5.2 (Inhomogeneous Short Integer Solution (ISIS$_{n,q,\beta,m}$)). *Given m random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$ composing the matrix A, and some $\mathbf{u} \in \mathbb{Z}_q^n$, find a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^m$ satisfying $||z|| \leq \beta$ such that*

$$A\mathbf{z} = \sum_i^n \mathbf{a}_i \cdot z_i = \mathbf{u} \mod q$$

This formulation will be helpful in Chapter 2, when talking about trapdoor functions. In fact, we will present a trapdoor for the inhomogeneheous SIS problem.

1.5.2 The Learning With Errors Problem

While the SIS problem is the foundation for signatures, the *Learning With Errors* problem has the same role for encryption. Informally, the problem asks to recover a secret vector given *many random noisy inner products* involving it.

The idea of the problem is very similar to the *Bounded Distance Decoding* one. However, while in the BDD problem we started with a lattice point and perturbed it with some noise, here we *compute* a lattice point having some randomness, and *then* perturb it with noise. From a mathematical point of view, though, one can imagine them being pretty much the same.

Definition 1.5.3 (Learning With Errors (LWE) instance). *Starting from a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ and choosing at random $\mathbf{a} \in \mathbb{Z}_q^n, e \leftarrow \chi$ (with χ a distribution over $\mathbb{Z}$, most often Gaussian), the corresponding Learning With Errors instance is*

$$(\mathbf{a}, b) = (\mathbf{a}, \quad \mathbf{s} \cdot \mathbf{a} + e \mod q)$$

There are two versions of the LWE problem: *search*, which consists in recovering the secret vector $\mathbf{s}$ given some LWE samples; *decision*, which consists in distinguishing between real LWE instances and uniformly random ones. Both of them are parametrized by n, q, χ, m, although m is of secondary importance as for SIS.

Definition 1.5.4 (Search-LWE$_{n,q,\chi,m}$). *Given m independent Learning With Errors instances $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ for a uniformly random secret $\mathbf{s} \in \mathbb{Z}_q^n$ (fixed for all samples), find $\mathbf{s}$.*

Definition 1.5.5 (Decision-LWE$_{n,q,\chi,m}$). *Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, with each being either a LWE instance or a uniformly sampled from $\mathbb{Z}_q^n \times \mathbb{Z}_q$, distinguish which is the case.*

The samples can be more conveniently arranged in a matrix $A \in \mathbb{Z}_q^{n \times m}$ and vectors $\mathbf{b}, \mathbf{e}$, so that the problem may be rewritten in vector notation:

$$\mathbf{b} = A\mathbf{s} + \mathbf{e} \mod q$$

Remark 1.5.5. *Both versions of the problem become easy if the noise is removed. In fact, they can easily be solved via Gaussian elimination (although, in the Decision version, it is likely that no solution will exist).*

Remark 1.5.6. *The solution to a given LWE instance is always unique. This is opposite to what happens for SIS.*

Search-LWE can be seen as a *Bounded Distance Decoding* problem on a certain family of q-ary m-dimensional lattices. In fact, for samples in matrix A, the vector $\mathbf{b}$ is relatively close to exactly one vector in the lattice

$$\mathcal{L}(A) = \{A^t\mathbf{s} \mod q : \mathbf{s} \in \mathbb{Z}_q^n\}$$

Similarly to SIS, the hardness of *Learning With Errors* can be reduced to that of the *Shortest Vector Problem.*

Theorem 1.5.2. *For any $m = poly(n), q \leq 2^{poly(n)}$ and discretized Gaussian distribution χ, solving the decision-LWE$_{n,q,\chi,m}$ is at least as hard as quantumly solving GapSVP$_\gamma$ and SIVP$_\gamma$ on arbitrary n-dimensional lattices, for some $\gamma = O(n)$ [Pei09].*

The theorem is proved with a quantum polynomial-time reduction, hence transforming any algorithm (classical or quantum, regardless) for solving LWE into a quantum algorithm for lattice problems. To date, there are no known algorithms for GavSVP and SIVP that perform significantly better than classical ones.

However desirable a fully classical proof would be, it has not been devised yet. There has been work in that direction, but it only partially achieved such a goal.

Chapter 2

Trapdoor Functions on Lattices

Informally speaking, a trapdoor function is a function easy to compute in one direction, but very difficult to invert, as illustrated in Figure 2.1. However, the inversion becomes easy when one knows some particular information.

The most straightforward analogy is that of a padlock: it is very easy to lock it, but very difficult to open it *unless* one has the key. In this concrete example, the key corresponds to the trapdoor.

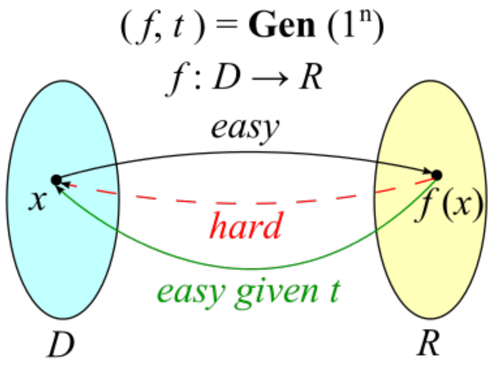

Figure 2.1: A trapdoor function.

Trapdoor functions are the foundation of asymmetric cryptography: the idea is that an instance of some hard problem can be shared publicly, while the corresponding information to efficiently solve it (i.e. the trapdoor) is kept secret. Then, the public information can be used to encrypt some data, while the secret key is needed to decrypt it. In a signature scheme, the secret key is needed to sign the data, while the public key is used to verify the signature.

In our work, we are concerned with trapdoor functions over lattices. It is worth mentioning that, even if our work is centered around homomorphic schemes, trapdoor functions are detached from the homomorphic notion. In fact, the homomorphic requirement is a feature that is put on top of the encryption/signature scheme, but does not have anything to do with trapdoor functions themselves.

The first trapdoor constructions on lattices are relatively old and date as back as [Ajt96] and [Ajt99]. However, we will stick to the more modern formulations given by [MP12], which is mathematically and computationally more versed.

2.1 Trapdoor for Learning With Errors

Let us consider some integer $q = poly(n)$, with n being the main security parameter, and the function

$$h_A(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t A + \mathbf{e}^t \mod q.$$

Notice how the function h_A has a straightforward relationship with the *Learning With Errors* problem in vector form, that we have already defined as a *hard problem* in Section 1.5.2.

What we will see is that, by carefully crafting the matrix A that parametrizes the function (i.e. parametrizes the LWE instance at hand), it is possible to generate another matrix R that makes solving the problem easy. In other words, if we *craft the problem* in a certain way, then we also know how to solve it efficiently.

Of course, only the party who generated the problem in the first place can derive the needed information to solve it. To any other party, the matrix A thus generated, even if *carefully crafted*, will not have any particular structure. That means that any other party will not be able to infer how to easily solve the problem by just seeing the *public* information.

Remark 2.1.1. *We will see that the trapdoor definition we are going to give is not only applicable to LWE, but can also be adapted to SIS problems.*

2.1.1 G-trapdoors

The secret information that makes up the trapdoor depends on the specific instance of the problem A. Moreover, it relies on the existence of a so-called *primitive matrix* G, the formal definition and discussion of which is deferred to Section 2.3. In fact, we prefer to give an overview of the trapdoor workings first, and dive into the detailed inner workings later. For now, it is enough to think of G as being a full rank matrix for which it exists an *algorithm* that can invert its transformation.

Definition 2.1.1 (G-trapdoor for A). *Let $A \in \mathbb{Z}_q^{n\times m}$ and $G \in \mathbb{Z}_q^{n\times w}$ a primitive matrix, with $m \geq \omega \geq n$. A G-trapdoor for A is a matrix $T \in \mathbb{Z}_q^{(m-\omega)\times\omega}$ such that $A \begin{bmatrix} T \\ I \end{bmatrix} = HG$ where $H \in \mathbb{Z}^{n\times n}$ is invertible (called the label or tag of the trapdoor).*
The quality of the trapdoor is measured by its largest singular value $s_1(T)$.

As already remarked, this notion of trapdoor is new. Previously, a trapdoor for lattices would be made of a short basis of $\mathcal{L}^\perp(A)$, but that method was deemed slow and complicated. It can be shown that this (modern) trapdoor definition *also* allows to retrieve a short basis of $\mathcal{L}^\perp(A)$. This proves that the G-trapdoor definition by [MP12] is *at least as good* as the old idea (yet, also significantly more powerful).

Remark 2.1.2. *If T is a trapdoor for A, then it can be used to build a trapdoor for any extension of $[A \mid B]$ by padding T with zero rows. This leaves $s_1(T)$ unchanged.*

Remark 2.1.3. *While the tag H does not seem to play a primary role in the definition of a G-trapdoor, it is actually fundamental. In fact, as we will see in Section 2.4, the trapdoor is built starting from H, not from T. In some sense, one can think of T as the trapdoor, while of H as the seed that generates it.*

For now, suppose to have a lattice defined by A alongside with its trapdoor (we will cover how trapdoors are generated in Section 2.4). Also, suppose that, while the function $h_A(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t A + \mathbf{e}^t$ is difficult to invert, the function $h_G(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t G + \mathbf{e}^t$ is easy instead. In other words, suppose that it exists an *oracle* for the inversion of h_G. This last hypothesis may seem arbitrary, but Section 2.3 provides an example of a matrix G for which this assumption holds.

To invert an instance of the *Learning With Errors* problem, the idea is to use the G-trapdoor T to transform the LWE instance based on A on an instance based on G, and then rely on the G-oracle to invert this last instance and recover the secret values $\mathbf{s}$ and $\mathbf{e}$.

Let us now see how the procedure works in detail.

Start from a lattice point $\mathbf{b}^t = h_A(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t A + \mathbf{e}^t \mod q$, where $\mathbf{s} \in \mathbb{Z}^n$ and $\mathbf{e} \in \mathbb{Z}^m$. Then define $\hat{\mathbf{b}}$ as:

$$\begin{aligned} \hat{\mathbf{b}}^t = \mathbf{b}^t \begin{bmatrix} T \\ I \end{bmatrix} = (\mathbf{s}^t A + \mathbf{e}^t) \begin{bmatrix} T \\ I \end{bmatrix} &= \mathbf{s}^t A \begin{bmatrix} T \\ I \end{bmatrix} + \underbrace{\mathbf{e}^t \begin{bmatrix} T \\ I \end{bmatrix}}_{\text{error}} = \\ &= \mathbf{s}^t H G + \hat{\mathbf{e}}^t = \hat{\mathbf{s}}^t G + \hat{\mathbf{e}}^t = \\ &= h_G(\hat{\mathbf{s}}, \hat{\mathbf{e}}) \end{aligned} \tag{2.1}$$

This allowed us to turn an h_A instance into an h_G instance! At this point, we can call the G-oracle on $\hat{\mathbf{b}}$ to obtain the values $(\hat{\mathbf{s}}, \hat{\mathbf{e}}) = \mathcal{O}(\hat{\mathbf{b}})$.

Finally, the original secret and error values can be recovered as

$$\mathbf{s}^t = H^{-t} \hat{\mathbf{s}}$$
$$\mathbf{e}^t = \mathbf{b}^t - \mathbf{s}^t A.$$

Let us stop to notice that the algorithm relied on the ability to discard the error value $\hat{\mathbf{e}}$ in Equation 2.1, thus leaving the term $\hat{\mathbf{s}}^t G$ alone. This can be done as long as there is a systematic and efficient way to get rid of the error term. This is the case as long as $\hat{\mathbf{e}} \in \mathcal{P}_{1/2}(B^{-t})$. There is a very nice algorithm to achieve this goal, based on Babai's *Nearest Hyperplane Algorithm* [Kap04].

What we still have not tackled is how the oracle $\mathcal{O}(\hat{\mathbf{b}})$ works, i.e. how it is possible to invert the transformation G. We will deal with this matter in Section 2.3.

In the end, then, the role of the trapdoor is to allow reducing an LWE instance on A, which is difficult, to an LWE instance on G, for which there are computationally efficient methods. Finally, notice how *anybody* could solve the instance on G - the role of the trapdoor is just to *get to it*.

2.2 Trapdoor for Short Integer Solution

Let us now see how the G-trapdoor notion can be used as a trapdoor function for SIS problem instances as well. We will consider the more generic case of the *Inhomogeneous-Short Integer Solution* problem.

SamPre($A, \mathbf{u}, T$) algorithm. The problem at hand is the following: given a lattice point $\mathbf{u} \in \mathbb{Z}^n$, find another element $\mathbf{x} \in \mathbb{Z}^m$ such that $f_A(\mathbf{x}) = A\mathbf{x} = \mathbf{u} \mod q$. This is also referred to as *preimage sampling.*

First, we need to find an element $\mathbf{z} \in \mathbb{Z}^\omega$ such that

$$f_G(\mathbf{z}) = G\mathbf{z} = H^{-1}(\mathbf{u} - A\mathbf{p}) \mod q$$

where $\mathbf{p} \in \mathbb{Z}^m$ is an apt perturbation vector with covariance matrix dependent on the trapdoor T [SD18]. This is a linear system with no additional constraints and can be solved efficiently.

Then, we define the vector $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{p} + \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z}$$

which satisfies the equality $A\mathbf{x} = \mathbf{u}$. In fact,

$$\begin{aligned} A\mathbf{x} = A\left(\mathbf{p} + \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z}\right) &= A\mathbf{p} + A \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z} = \\ &= A\mathbf{p} + HG\mathbf{z} = A\mathbf{p} + HH^{-1}(\mathbf{u} - A\mathbf{p}) = \mathbf{u} \end{aligned}$$

In the end, being able to craft a vector $\mathbf{z}$ as above and owning the G-trapdoor T for A allows the inversion of an SIS instance over A.

Remark 2.2.1. *Of course, the inversion of an instance of the homogeneous-SIS (i.e. the case $f_A(\mathbf{x}) = A\mathbf{x} = \mathbf{0} \mod q$) stems directly from the discussion above. Setting $\mathbf{u} = \mathbf{0}$ will leave everything else working the same.*

2.3 Primitive matrices and inverting h_G

The whole trapdoor construction we have built so far relies on the ability to efficiently invert h_G. In this section, we show how this can be done.

We will start by providing more details on the structure of the matrix G. In the previous sections, we had assumed (rather vaguely) that it must be possible to invert the transformation of G. We will now show how this can be achieved.

Notice how we never requested G to be an *invertible* matrix. In fact, this is not needed: we only need to be able to revert its action. This may seem odd at first, but makes more sense if we think that G has the effect of re-constructing an integer given its binary representation. With this in mind, it is clear that we do not need to find the explicit matrix inverse of G to invert its action: we simply need a (very simple!) algorithm to extract the binary representation of an integer.

As we have seen, the matrix G is a pivot point for trapdoors both for LWE and SIS problem instances. The requirement it must satisfy is to be *primitive.* Intuitively, one can think of a matrix being primitive when full rank - i.e. when it generates the whole space. We now provide precise definitions:

Definition 2.3.1 (Primitive vector). *A vector* $\mathbf{g} \in \mathbb{Z}_q^k$ *is primitive if its components satisfy* $\gcd(g_1, \cdots, g_k, q) = 1$.

Definition 2.3.2 (Primitive matrix). *A matrix* $G \in \mathbb{Z}_q^{n\times m}$ *is primitive if its columns generate all of* $\mathbb{Z}_q^n$. *In other words, G is primitive if* $G \cdot \mathbb{Z}^m = \mathbb{Z}^n$.

Primitive matrices are particularly important because solving instances of the *Learning With Errors* problems based on them (what we have referred to as 'inverting the function h_G') can be done efficiently. This is the core of the next important result, and we will spend the rest of this section going through its proof.

Theorem 2.3.1 (Inversion of h_G). *For any integers* $q \geq 2, n \geq 1, k = \lceil \log_2 q \rceil$ *and* $m = nk$, *there exists a primitive matrix* $G \in \mathbb{Z}_q^{n\times m}$ *such that:*

1. *The lattice* $\mathcal{L}^\perp(G)$ *has a known (short) basis* $S \in \mathbb{Z}^{m\times m}$, *such that* $||S|| \leq max\{\sqrt{5}, \sqrt{k}\}$.
2. *Both G and S are sparse and thus require little storage capacity.*
3. *Inverting the function* $h_G(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t G + \mathbf{e}^t \mod q$ *can be performed in quasilinear time* $O(n \log^c n)$ *for any* $\mathbf{s} \in \mathbb{Z}_q^n$ *and* $\mathbf{e} \in \mathcal{P}_{1/2}(q \cdot S)$.

Proof. The proof is made of several steps. First, we show how to build the matrices G and S; then we provide concrete instantiations of them for which we show how to efficiently invert the function h_G.

For this proof, set $q = 2^k$ an exact power of two.

Building G and S. Start with a primitive vector $\mathbf{g} \in \mathbb{Z}_q^k$. First, notice that, being primitive, $\mathbf{g}$ defines a k-dimensional lattice $\mathcal{L}^\perp(\mathbf{g}^t)$. The lattice is also full-rank because $\mathbb{Z}^k/\mathcal{L}^\perp(\mathbf{g}^t)$ is in bijection with the set $\{\mathbf{g}^t\mathbf{x} \mod q,\ \mathbf{x} \in \mathbb{Z}^k\} = \mathbb{Z}_q$ through the first homomorphism theorem.

Denote with $S_k \in \mathbb{Z}^{k\times k}$ a basis of $\mathcal{L}^\perp(\mathbf{g}^t)$. Being full-rank, that gives us $|\det(S_k)| = q$.

The primitive vector $\mathbf{g}$ and the corresponding basis S_k are used to define the matrix G and basis S as

$$G = I_n \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n\times nk}, \quad S = I_n \otimes S_k \in \mathbb{Z}^{nk\times nk}$$

$$G = \begin{bmatrix} \cdots \mathbf{g}^t \cdots & & & \\ & \cdots \mathbf{g}^t \cdots & & \\ & & \vdots & \\ & & & \cdots \mathbf{g}^t \cdots \end{bmatrix}, \quad S = \begin{bmatrix} \cdots S_k \cdots & & & \\ & \cdots S_k \cdots & & \\ & & \vdots & \\ & & & \cdots S_k \cdots \end{bmatrix}$$

In other words, G and S are built from the identity matrix of size n in which every diagonal (scalar) entry is replaced with a copy of $\mathbf{g}^t$ and S_k respectively, stretching the dimensions accordingly. One consequence of this construction is that, being $\mathbf{g}$ primitive, G is a primitive matrix.

Furthermore, notice how the inversion of $h_G(\mathbf{s}, \mathbf{e})$ is easily parallelizable by performing the same operations n times in parallel for $h_{\mathbf{g}^t}(\mathbf{s}, \mathbf{e})$, since G simply consists of several copies of $\mathbf{g}^t$. Thus, what is needed in the end is an algorithm for efficient inversion of $h_{\mathbf{g}^t}(\mathbf{s}, \mathbf{e})$. We will achieve this for a specific $\mathbf{g}^t$.

Concrete instantiations. Consider the primitive vector

$$\mathbf{g}^t = (1, 2, 4, \cdots, 2^{k-1}) \in \mathbb{Z}_q^{1\times k} \quad k = \lceil \log_2 q \rceil$$

and notice how $\mathbf{g}^t\mathbf{x} \in \mathbb{Z}_q$ for a binary $\mathbf{x} \in \{0,1\}^k$ is just its integer representation. Intuitively, that is what will make the inversion of $g_{\mathbf{g}^t}$ easy: the idea is that we only need to find the binary representation of a number in $\mathbb{Z}^k$.

Then define the matrix S_k as

$$S_k = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k\times k}$$

and remark that it is a basis of $\mathcal{L}^\perp(\mathbf{g}^t)$ since $\mathbf{g}^t \cdot S_k = \mathbf{0} \mod q$ and $\det(S_k) = 2^k = q$.

Efficient inversion of $h_{\mathbf{g}^t}(\mathbf{s}, \mathbf{e})$. We now get to the core of the proof. We have seen that inverting h_G reduces to inverting $h_{\mathbf{g}^t}$ because of how we have built G. Thus, we now need to show how to invert $h_{\mathbf{g}^t}$. Notice that the size of s needs to change accordingly, depending on whether we are considering the case of $\mathbf{g}$ or of G. In fact, with $\mathbf{g}$, s is a scalar; while with G, $\mathbf{s}$ is a vector.

Suppose to be given the vector

$$\begin{aligned} \mathbf{b}^t &= (b_0, b_1, \cdots, b_{k-1}) = \\ &= s \cdot \mathbf{g}^t + \mathbf{e}^t = (s + e_0, 2s + e_1, \cdots, 2^{k-1}s + e_{k-1}) \mod q, \end{aligned}$$

where $\mathbf{e} \in \mathbb{Z}^k$ is a *short* error vector and $s \in \mathbb{Z}_q$ is a secret scalar element. The goal is to recover the integer s.

There is an iterative algorithm that works by recovering the binary digits $s_0, s_1, \cdots, s_{k-1} \in \{0,1\}$ of $s \in \mathbb{Z}_q$, from the least to the most significant digit, in the following manner.

First, consider the least significant digit:

$$\begin{aligned} b_{k-1} = 2^{k-1}s + e_{k-1} &= \frac{q}{2}(s_0 \ s_1 \ \cdots \ s_{k-1}) + e_{k-1} \mod q \\ &= \frac{q}{2}s_0 + 2^k(\cdots) + e_{k-1} \mod q \\ &= \frac{q}{2}s_0 + e_{k-1} \mod q \end{aligned}$$

because $\frac{q}{2}s_i = 0 \mod q \;\; \forall i \geq 1$.

This last step may require some thought. It relies on the fact that the elements s_i are actually the binary representation of s, so their position already encodes some power of 2. For example, s_0 and s_1, though being both bits, have different values. In fact,

$$s_0 \to s_0 \cdot 2^0$$
$$s_1 \to s_1 \cdot 2^1$$
$$\vdots$$
$$s_{k-1} \to s_{k-1} \cdot 2^{k-1}$$

so any digit s_i with $i \geq 1$ has already a factor 2^i. Multiplying that by $\frac{q}{2} = 2^{k-1}$ yields a result that is multiple of $2^k = q$, thus being null modulo q.

Going forward with the algorithm, the next step is to determine whether b_{k-1} is closer to 0 or to $\frac{q}{2} \mod q$. The noise e_{k-1} is irrelevant as long as it is within $[-\frac{q}{4}, \frac{q}{4})$, which is a condition that needs to be met. This gives the value $\frac{q}{2}s_0$, from which it is trivial to recover the bit s_0.

Then, to recover the second digit s_1, look at

$$b_{k-2} = 2^{k-2}s + e_{k-2} = 2^{k-1}s_1 + s^{k-2}s_0 + e_{k-2} \mod q$$

and subtract $s^{k-2}s_0$ from it. Again, testing proximity to 0 or $\dfrac{q}{2}$ will yield the value of s_1. A similar procedure can be iterated to recover all binary digits of s.

As already mentioned, this algorithms successfully recovers all the s_i elements as long as $\mathbf{e} \in \mathcal{P}_{1/2}(q \cdot I_k/2)$. This allows to efficiently invert the function $h_{\mathbf{g}^t}(\mathbf{s}, \mathbf{e})$. $\square$

Remark 2.3.1. *The algorithm above is a specialized version of the Babai's* Nearest Hyperplane Algorithm *[Kap04].*

Remark 2.3.2. *Here we have set $q = 2^k$, but it is possible to use a modulus q that is not a power of two with slight adaptations. For details on arbitrary moduli, refer to [MP12].*

2.4 On the practical construction of trapdoors

Up to now, we have shown how trapdoor functions *work*, but we have not provided any methods to *build* them. In fact, we worked under the implicit hypothesis that, given a matrix A instance of a lattice hard problem (SIS or LWE), *it existed* a G-trapdoor for it that allowed easy resolution of the problem. We are now going to show *how* these trapdoors can be generated, as detailed in [MP12].

The high-level idea is to define the trapdoor matrix T first, and then *use that to generate a lattice* on which the trapdoor works as seen in previous sections.

Before starting, we need to state a result that will be needed throughout the construction.

Proposition 2.4.1. *Let $A \in \mathbb{Z}_q^{n\times m}$ be an arbitrary matrix and $S \in \mathbb{Z}^{m\times m}$ be any basis of $\mathcal{L}^{\perp}(A)$. Then, for any unimodular matrix $K \in \mathbb{Z}^{m\times m}$, we have that*

$$K \cdot \mathcal{L}^{\perp}(A) = \mathcal{L}^{\perp}(A \cdot K^{-1}),$$

with $K \cdot S$ as a basis. □

TrapGen$(1^n, 1^m, q)$ algorithm. We now show how to generate a lattice together with its trapdoor.

First, start from a random matrix $\bar{A} \in \mathbb{Z}_q^{n\times \bar{m}}$. Then, using the primitive matrix $G \in \mathbb{Z}_q^{n\times \omega}$, define the semi-random matrix

$$A' = [\bar{A} \mid HG] \in \mathbb{Z}_q^{n\times m},$$

where $m = \bar{m} + \omega$ and $H \in \mathbb{Z}^{n\times n}$ is the desired trapdoor tag. H can be random as well.

To continue, draw another matrix $T \in \mathbb{Z}^{\bar{m}\times \omega}$ according to some distribution $\mathcal{D}$ and use it to build the transformation matrix

$$K = \begin{bmatrix} I_{\bar{m}} & T \\ 0_{\omega} & I_{\bar{m}} \end{bmatrix} \in \mathbb{Z}^{m\times m}.$$

We would like to apply K to $\mathcal{L}^{\perp}(A')$. Since K is unimodular and its inverse is

$$K^{-1} = \begin{bmatrix} I_{\bar{m}} & -T \\ 0_{\omega} & I_{\bar{m}} \end{bmatrix} \in \mathbb{Z}^{m\times m},$$

using the proposition above we have that $K \cdot \mathcal{L}^{\perp}(A') = \mathcal{L}^{\perp}(A' \cdot K^{-1})$. Finally, output the result matrix A as

$$A = A \cdot K^{-1} = [\bar{A} \mid HG - \bar{A}T] \in \mathbb{Z}_q^{n\times m},$$

which represents the lattice (i.e. the public key) on which the trapdoor T works.

Moreover, the distribution of such a built A is close to uniform as long as the distribution of $[\bar{A} \mid 0] \cdot K^{-1} = [\bar{A} \mid -\bar{A}T]$ is such. This means that the above is a sound way of generating a lattice instance *alongside* with its trapdoor.

2.5 Trapdoor functions on polynomial rings

This section extends the trapdoor notion we have given for integer-modulo rings to polynomial rings drawing from [Ber+18]. This will be needed for the signature scheme based on polynomial rings that will be described in Section 3.3.

We first need to define the polynomial ring-version of the *Short Integer Solution problem* (Ring-SIS), which is the natural extension of the SIS problem. Throughout this section, let us denote with R the ring $\mathbb{Z}_q[X]/(X^d - 1)$, which consists of polynomials of degree at most $d - 1$ where each coefficient is reduced modulo q. In our study, we will focus on the case where d is a power of 2. More details on the mathematical background can be found in [SD18].

Definition 2.5.1 (Ring-SIS). *Given a vector of polynomials* $\mathbf{a} \in R^k$ *(or, which is the samek random polynomials* $a_i \in R$*), find a non-zero integer vector* $\mathbf{z} \in \{0,1\}^k$ *such that*

$$\mathbf{a}^t\mathbf{z} = \sum_i^n a_i z_i = 0 \in R$$

We may then give the notion of trapdoor function over polynomial rings.

Definition 2.5.2 (G-trapdoor on rings). *Let* $\mathbf{a} \in R^m$ *and* $\mathbf{g} \in R^k$ *with* $k = \lceil \log_2 q \rceil$, $m > k$. *A* $\mathbf{g}$*-trapdoor for* $\mathbf{a}$ *is a matrix of small polynomials* $T \in R^{(m-k)\times k}$ *such that* $\mathbf{a}^t \begin{bmatrix} T \\ I_k \end{bmatrix} = h\mathbf{g}^t$ *for an invertible element* $h \in R$ *(referred to as the tag).*

We remark that an instance analogous to that of integer moduli for the primitive vector $\mathbf{g}$ is that of constant polynomials $\mathbf{g}^t = (1, 2, 4, \cdots, 2^{k-1}) \in R^k$.

Given this definition, it is easy to verify that the trapdoor for SIS already discussed keeps working [Ber+18], as long as the sizes are properly adjusted.

In particular, to find an element $\mathbf{x} \in R^m$ such that $f_{\mathbf{a}^t}(\mathbf{x}) = \mathbf{a}^t\mathbf{x} = u$, with $u \in R$, we first need to find an element $\mathbf{z} \in R^m$ such that

$$f_{\mathbf{g}^t}(\mathbf{z}) = \mathbf{g}^t\mathbf{z} = H^{-1}(u - \mathbf{a}^t\mathbf{p})$$

where $\mathbf{p} \in R^m$ is an apt perturbation vector with covariance matrix dependent on the trapdoor T [SD18]. This is a linear system with no additional constraints and can be solved efficiently.

Then, we define the vector $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{p} + \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z}$$

which satisfies the equality $\mathbf{a}^t\mathbf{x} = u$. In fact,

$$\begin{aligned} \mathbf{a}^t\mathbf{x} = \mathbf{a}^t \left(\mathbf{p} + \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z}\right) &= \mathbf{a}^t\mathbf{p} + \mathbf{a}^t \begin{bmatrix} T \\ I \end{bmatrix} \mathbf{z} = \\ &= \mathbf{a}^t\mathbf{p} + h\mathbf{g}^t\mathbf{z} = \mathbf{a}^t\mathbf{p} + hh^{-1}(u - \mathbf{a}^t\mathbf{p}) = u \end{aligned}$$

which is the same result we had derived for G-trapdoors on the ring of integer moduli: here as well, the $\mathbf{g}$-trapdoor allows reducing a SIS-instance on $\mathbf{a}$ to a SIS-instance on $\mathbf{g}$, which is easily solvable.

Chapter 3

Homomorphic Signatures

Imagine that Alice owns a large data set, over which she would like to perform some computation. In a homomorphic signature scheme, Alice signs the data set with her secret key and uploads the signed data to an untrusted server. The server then performs the computation modeled by the function g to obtain the result $y = g(x)$ over the signed data.

Alongside the result y, the server also computes a signature $\sigma_{g,y}$ certifying that y is the correct result for $g(x)$. The signature should be short - at any rate, it must be independent of the size of x. Using Alice's public verification key, anybody can verify the tuple $(g, y, \sigma_{g,y})$ without having to retrieve all the data set x nor to run the computation $g(x)$ on their own again.

The signature $\sigma_{g,y}$ is a *homomorphic signature*, where *homomorphic* has the same meaning as the mathematical definition: '*mapping of a mathematical structure into another one in such a way that the result obtained by applying the operations to elements of the first structure is mapped onto the result obtained by applying the corresponding operations to their respective images in the second one*' [Mer]. In our case, the *operations* are represented by the function f, and the *mapping* is from the matrices $U_i \in \mathbb{Z}_q^{n\times n}$ to the matrices $V_i \in \mathbb{Z}_q^{n\times m}$.

Notice how the very idea of homomorphic signatures challenges the basic security requirements of traditional digital signatures. In fact, for a traditional signatures scheme we require that it should be computationally infeasible to generate a valid signature for a party without knowing that party's private key. Here, we *need* to be able to generate a valid signature on *some data* (i.e. results of computation, like $g(x)$) *without* knowing the secret key. What we require, though, is that it must be computationally infeasible to forge a valid signature σ' for a result $y' \neq g(x)$. In other words, the security requirement is that *it must not be possible to cheat on the signature of the result*: if the provided result is validly signed, then it must be the *correct* result.

In the next sections we present the signature scheme devised by Gorbunov, Vaikuntanathan and Wichs [GVW15]. As already mentioned, it relies on the *Short Integer Solution* hard problem on lattices. The scheme presents several limitations and possible improvements, but it is also the first homomorphic signature scheme able to evaluate arbitrary arithmetic circuits over signed data.

We start with some basic definitions and then move on to the formal notions.

Definition 3.0.1. *A signature scheme is said to be leveled homomorphic if it can only evaluate circuits of fixed depth d over the signed data, with d being function of the security parameter. In particular, each signature σ_i comes with a noise level β_i: if, combining the signatures into the result signature σ, the noise level grows to exceed a given threshold β^*, then the signature σ is no longer guaranteed to be correct.*

Definition 3.0.2. *A signature scheme is said to be fully homomorphic if it supports the evaluation of any arithmetic circuit (albeit possibly being of fixed size, i.e. leveled). In other words, there is no limitation on the "richness" of the function to be evaluated, although there may be on its complexity.*

Let us remark that, to date, no (*non-leveled*) fully homomorphic signature scheme has been devised yet. The state of the art still lies in *leveled* schemes, such as [GVW15]. On the other hand, a great breakthrough was the invention of a fully homomorphic *encryption* scheme by [Gen09].

3.1 The scheme on integers

The signature scheme on integers relies on the trapdoor function for the *Short Integer Solution* problem. In this setup, signatures consist of matrices $V_i \in \mathbb{Z}_q^{n\times m}$, while the "signature secrets" are matrices $U_i \in \mathbb{Z}_q^{m\times m}$ with entries whose sizes are properly bounded (recall that it is exactly this requirement on the magnitude of the solution that makes SIS difficult to break). The SIS instance is encoded in a matrix $A \in \mathbb{Z}_q^{n\times m}$.

Throughout this section, let us denote with $x_i \in \{0,1\}$ the bits to be signed and $G \in \mathbb{Z}_q^{n\times m}$ a primitive matrix. Let us also denote with g the function we would like to evaluate on the data set $\{x_i\}$, expressed as an arithmetic circuit (being a Turing complete language, any function can be expressed as such).

We now detail the algorithms which make up the signature scheme.

- **prms** $\leftarrow$ **prmsGen**$(1^\lambda, 1^N)$**.** Gets the security parameter λ and the data-size bound N. Generates the parameters n, m and q (all together referred to as **prms**). The values should be set so that the corresponding $SIS_{n,q,\beta,m}$ instance both is difficult and admits solution. Also notice how the parameters define the message space $\mathcal{X}$.

  Then, a set of matrices $V_1, \cdots, V_N \in \mathbb{Z}_q^{n\times m}$ is uniformly sampled.

- **(*pk*, *sk*)** $\leftarrow$ **KeyGen(**1^λ**, prms).** Here the public key *pk* and the secret key *sk* are generated. With the methods detailed in Section 2.4, a matrix $A \in \mathbb{Z}_q^{n\times m}$ is generated together with the relative G-trapdoor $T \in \mathbb{Z}^{(m-k)\times k}$. A will serve as public verification key, while T as secret key.

- $\sigma_1, \cdots, \sigma_N \leftarrow$ **Sign**$_{sk}(x_1, \cdots, x_N)$**.** The signature function is in the form

$$\sigma_i = f_{A,x_i}(U_i) = AU_i + x_iG = V_i$$

and the trapdoor is used to retrieve the value of $U_i \in \mathbb{Z}_q^{m\times m}$ by running it against a given $V_i - xG$ as described in Section 2.2. The matrix U_i is kept secret and can only be obtained with knowledge of the trapdoor.

- $\sigma^* \leftarrow \mathbf{SignEval}_{prms}(g, ((x_1, \sigma_1), \cdots, (x_N, \sigma_N)))$ Homomorphically computes the signature σ^* for the computation of g on the signed data set $\{(x_i, \sigma_i)\}$. In our use case, this is done by the cloud server acting on the V_i matrices. Refer to Section 3.1.2 for how the signature is homomorphically computed in practice.
- $\alpha_g \leftarrow \mathbf{Process}_{prms}(g)$ Homomorphically computes the verification key α_g for function g from the public parameters. In our case, this is done by the user who signed the data set acting on the U_i matrices. Refer to Section 3.1.2 for how the key is homomorphically computed in practice.
- $\mathbf{Verify}_{pk}(\alpha_g, y, \sigma)$ Verify that y really is the result of $g(x_1, \cdots, x_N)$ by checking the signature σ against α_g. In particular, it checks whether

  $$f_{A,y}(\sigma^*) = A\sigma^* + yG \overset{?}{=} \alpha_g$$

  or, with matrix notation, whether

  $$f_{A,y}(U^*) = AU^* + yG \overset{?}{=} V^*$$

Notice how each individual bit x_i requires a full matrix U_i to be signed (size n^2), and results in a matrix V_i as signature (size $(n \times m)^2$).

Remark 3.1.1. *It is useful to think the **Process** algorithm as a "pre-processing" of the function g. The complexity of this step depends on the circuit of g and can be significant. On the other hand, **Verify** is very quick and its complexity does not depend on g.*

*The real verification routine requires both **Process** and **Verify** (sometimes referred to as **Verify***), but we distinguish between the two as **Process** can be performed offline, prior to seeing the result y. This allows to speed up the verification phase significantly!*

Remark 3.1.2. *We said that the matrices V_i are randomly sampled by **prmsGen**. However, we can relax this requirement. In fact, since the V_i are to all effects public parameters, they can also be set once for all, for all users of the scheme. For example, they could be embedded into the signature software. Then, each user retrieves the corresponding secret U_i using their own trapdoor (which must be different for each of them, of course). As we will see, Theorem 3.1.1 guarantees that there is no exploitable relationship between a V_i and its corresponding U_i, so fixing the V_i does not affect security.*

Evaluation correctness. Now we port the classical correctness requirement of digital signatures in the homomorphic setting.

We require that for any choice of parameters **prms**, keys *(pk, sk)*, message $(x_1, \cdots, x_N)$, signatures $(\sigma_1, \cdots, \sigma_N)$ and function $g : \mathcal{X}^N \to \mathcal{X}$, we have:

$$\mathbf{Verify}^*_{pk}(g, g(x_1, \cdots, x_N), \sigma^*) = \mathbf{accept},$$

where $\sigma^* \leftarrow \mathbf{SignEval}_{prms}(g, ((x_1, \sigma_1), \cdots, (x_N, \sigma_N)))$.

3.1.1 Statistical indistinguishability

There is one last piece of the puzzle that is still missing, over which we have not paid attention up to now. That is: what is the probability distribution of the U_i and V_i, and do the trapdoor workings have some kind of predictable outcome?

In other words, since SIS is a hard problem, we know that it is not possible to infer the matrices U_i if given only the corresponding V_i. However, does this still hold if we know that the U_i were obtained *through* a trapdoor? *Does the trapdoor give away any information?*

Furthermore, we have seen that to generate the matrix A that defines the SIS instance, we start from its trapdoor T. For this reason, another concern is: does the A thus generated look random? Or maybe it carries some information about the trapdoor?

We will now provide definitions that help to formalize these questions and provide important results that guarantee that our construction is sound.

Definition 3.1.1. *For discrete random variables X, Y with support $\mathcal{X}, \mathcal{Y}$ respectively, their statistical distance is*

$$SD(X,Y) = \frac{1}{2} \sum_{u \in \mathcal{X} \cup \mathcal{Y}} |\mathbb{P}(X = u) - \mathbb{P}(Y = u)|.$$

Definition 3.1.2. *Two ensembles of random variables $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ are statistically close, denoted by $X \overset{stat}{\approx} Y$, if $SD(X,Y)$ is a negligible function of λ.*

The next theorem is taken from [GVW15], but it has been an incremental result built from different sources ([Ajt99; GPV08; AP11; MP12]).

Theorem 3.1.1. *Given integers $n \geq 1, q \geq 2$, there exists some $m^* = m^*(n,q) = O(n \log q)$ and $\beta = \beta(n,q) = O(n\sqrt{\log q})$ such that, for all $m \geq m^*$, we have the statistical indistinguishability requirements:*

$$A \overset{stat}{\approx} A', \qquad (A,T,U,V) \overset{stat}{\approx} (A,T,U',V')$$

*where $(A,T) \leftarrow$ **TrapGen**$(1^n, 1^m, q)$, A' is uniformly sampled from $\mathbb{Z}_q^{m \times n}$ and $U \in \mathbb{Z}_q^{n \times n}$ is uniformly sampled satisfying the constraint $||U||_\infty \leq \beta$, $V = AU$, V' is randomly sampled from $\mathbb{Z}_q^{n \times m}$, $U' \leftarrow$ **SamPre**(A, V', T).*

*Statistical distances are negligible in n. Moreover, any $U' \leftarrow$ **SamPre**(A, V', T) always satisfies $AU' = V'$ and $||U'||_\infty \leq \beta$.*

In other words, this important theorem states that we can be sure of two facts:

1. generating an instance A through **TrapGen** is not statistically different than picking one at random;
2. picking a random "signature secret" U and computing the corresponding "public signature" $V = AU$ is not statistically different than starting with a random public signature V' and going backwards to its U' using the trapdoor T and **SamPre**.

Taken together, they imply that the signature scheme and the trapdoor function we have built up to now are secure.

3.1.2 Homomorphic operations

The algorithms **SignEval** and **Process** are responsible of processing the signatures V_i and the 'signatures secrets' U_i. We now provide details on how exactly these manipulations happen.

We consider the homomorphic evaluation of certain base functions that can be used to build an arbitrary arithmetic circuit. These basic *building blocks* allow to evaluate any function g.

As we remarked earlier, we must take care about the depth of the circuit we want to evaluate, as the scheme is *leveled.* For this reason, in defining the basic operations, we must keep in mind that each signature has some *noise* that propagates throughout the processing and, at some point, may make the final signature incorrect.

Definition 3.1.3. *The noise of a 'signature secret' U_i is $\beta_i = ||U_i||_\infty \in \mathbb{R}$.*

Each 'signature secret' U_i starts with some (low) noise β_i. If the noise level β^* of the result U^* exceeds some threshold (defined by **prms**), then the evaluation correctness as defined in 3.1 needs not hold. This is the real *limit on the set of functions whole evaluation is supported* by the signature scheme: the ones such that the noise level β^* stays below the threshold. Understanding the error-growth rate is thus central in the next discussion.

Let us now go through the basic operations definitions. For each of them, we provide the required computation for algorithms **SignEval** and **Process**.

- **Addition gate:** $g(x_1, x_2) = x_1 + x_2$.

  $$U^* = U_1 + U_2 \qquad V^* = V_1 + V_2$$

  The noise of the result is $\beta^* \leq \beta_1 + \beta_2$.

- **Multiplication gate:** $g(x_1, x_2) = x_1 \cdot x_2$.

  $$U^* = x_2 \cdot U_1 + U_2\ G^{-1}(V_1) \qquad V^* = V_2 \cdot G^{-1}(V_1)$$

  The noise of the result is $\beta^* \leq |x_2|\beta_1 + m\beta_2$. Notice how the error growth is asymmetric with respect to the x_i values.

- **Addition-with-constant gate:** $g(x) = x + c$, with $c \in \mathbb{Z}_q$.

  $$U^* = U_1 \qquad V^* = V_1 + c \cdot G$$

  The noise of the result is $\beta^* = \beta_1$, resulting in no change from the start.

- **Multiplication-by-constant gate:** $g(x) = x \cdot c$, with $c \in \mathbb{Z}_q$.

  $$U^* = c \cdot U \qquad V^* = c \cdot V$$

  The noise of the result is $\beta^* = |c|\beta$. Notice how this method requires a small c. However, there is also an alternative way of computing this gate:

  $$U^* = U\ G^{-1}(c \cdot G) \qquad V^* = V\ G^{-1}(c \cdot G)$$

With the noise of the result being $\beta^* = m\beta$, which is independent of the value of c. The most fit gate instantiation can be chosen depending on which between $|c|$ and m is bigger to achieve the smallest error growth.

It is straightforward enough to check that, if the inputs U_i, V_i satisfy $V_i = f_{A,x_i}(U_i)$, then the homomorphic evaluation procedures described above ensure that $f_{A,g(x_1,\cdots,x_N)}(U^*) = V^*$. The gates can be composed to build an arbitrary function g expressed as an arithmetic circuit. The only real limitation is the noise growth.

3.2 A note on the hopes for Fully Homomorphic Signatures

The main limitation of the current construction is that verifying the correctness of the computation takes Alice roughly as much time as the computation of $g(x)$ itself. However, what she gains is that she does not have to store the data set long term, but can do only with the signatures.

To us, this limitation makes intuitive sense, and it is worth comparing it with real life. In fact, if one wants to judge the work of someone else, they cannot just look at it without any preparatory work. Instead, they have to have spent (at least) *a comparable amount of time* studying/learning the content to be able to evaluate the work.

For example, a good musician is required to evaluate the performance of Beethoven's Ninth Symphony by some orchestra. Notice how anybody with some musical knowledge could evaluate whether what is being played *makes sense* (for instance, whether it actually *is* the Ninth Symphony and not something else). On the other hand, evaluating the perfection of performance is something entirely different and requires years of study in the music field and in-depth knowledge of the particular symphony itself.

That is why it looks like hoping to devise a homomorphic scheme in which the verification time is significantly shorter than the computation time would be against what is rightful to hope. It may be easy to judge whether the result makes sense (for example, it is not a letter if we expected an integer), but is difficult if we want to evaluate perfect correctness.

However, there is one more caveat. If Alice has to verify the result of the same function g over two different data sets, then the verification cost is basically the same (*amortized verification*). Again, this makes sense: when one is skilled enough to evaluate the performance of the Ninth Symphony by the *Berlin Philharmonic*, they are also skilled enough to evaluate the performance of the same piece by the *Vienna Philharmonic*, without having to undergo any significant further work other than going and *listening to* the performance.

So, although it does not seem feasible to devise a scheme that *guarantees* the correctness of the result and in which the verification complexity is significantly less than the computation complexity, not all hope for improvements is lost. In fact, it may be possible to obtain a scheme in which verification is faster, but the correctness is only *probabilistically guaranteed.*

Back to our music analogy, we can imagine the evaluator *listening to a handful* of minutes of the Symphony and evaluate the whole performance from the little he has heard. However, the orchestra has no idea at what time the evaluator will show up, and for how long they will listen. Clearly, if the orchestra makes a mistake in those few minutes, the performance is not perfect; on the other hand, if what they hear is flawless, then there is *some probability* that the whole play is perfect.

Similarly, the scheme may be tweaked to *only partially check the signature result*, thus assigning a *probabilistic measure of correctness.* As a rough example, we may think of not computing the homomorphic transformations over the U_i matrices wholly, but only calculating a few, randomly-placed entries. Then, if those entries are all correct, it is very *unlikely* (and it quickly gets more so as the number of checked entries increases, of course) that the result is wrong. After all, to cheat, the third party would need to guess several numbers in $\mathbb{Z}_q$, each having $1/q$ likelihood of coming up!

Another idea would be for the music evaluator to delegate another person to check for the quality of the performance, by giving them some precise and detailed features to look for when hearing the play. In the homomorphic scheme, this may translate in *looking for some specific features in the result*, some characteristics we know *a priori* that must be in the result. For example, we may know that the result must be a prime number, or must satisfy some constraint, or a relation with something much easier to check. In other words, we may be able to *reduce the correctness check to a few fundamental traits* that are very easy to check, but also provide some guarantee of correctness. This method seems much harder to model, though.

We did not develop any of these ideas further in this work, but we believe they are promising hints for further improvements.

3.3 The scheme on polynomial rings

Here, we show how to port the signature scheme from the integer-modulo rings $\mathbb{Z}_q$ to polynomial rings $\mathbb{Z}_q[X]/(X^d - 1)$ using the notion of $\mathbf{g}$-trapdoor over polynomial rings defined in Section 2.5, drawing inspiration from a similar approach for homomorphic encryption by [KGV16].

The core of the scheme is similar to the one on integers, with the fundamental difference that the elementary units are not integers in $\mathbb{Z}_q$ anymore, but rather polynomials of degree at most $d - 1$ with coefficients in $\mathbb{Z}_q$. The reason for such an interest is to obtain improvements in computation time and ability to sign more than one bit in a single shot. An analysis of the improvements over the original scheme can be found in Section 3.3.2.

In this setup, signatures consist of vectors of polynomials $\mathbf{v} \in R^k$, while the 'signature secrets' consist of matrices $U_i \in \mathbb{Z}_q^{m\times k}$ with entries of properly bounded magnitudes. In this context, this means that the entries of U_i should have coefficients in the set $\{-1, 0, 1\}$.

Throughout this section, denote with $x_i \in R_{\{-1,0,1\}}$ the *polynomials* to be signed, having coefficients in $\{-1, 0, 1\}$, and with $\mathbf{g} \in R^k$ a primitive vector. Also, let g be the function we would like to evaluate on the data set $\{x_i\}$, expressed as an arithmetic circuit.

We now go through the scheme algorithms.

- **prms** $\leftarrow$ **prmsGen**$(1^\lambda, 1^d)$. Gets the security parameter λ and the maximum degree d, which effectively acts as data-size bound. Generates the parameters n, m and q (**prms**). The values should be set so that the corresponding $Ring-SIS_{n,q,\beta,m}$ instance is difficult and admits solution. Here as well, the parameters define the message space $\mathcal{X}$.

  Then, a set of vectors $\mathbf{v}_1, \cdots, \mathbf{v}_N \in R^k$ is uniformly sampled.

- **(*pk, sk*)** $\leftarrow$ **KeyGen(**1^λ**, prms).** Generates the public key *pk* and the secret key *sk*. With the methods detailed in Section 2.4, a vector $\mathbf{a} \in R^m$ is generated together with the relative g-trapdoor $T \in \mathbb{Z}^{(m-k)\times k}$. A will serve as public verification key, while T as secret key.

- $\sigma_1, \cdots, \sigma_N \leftarrow$ **Sign**$_{sk}(x_1, \cdots, x_N)$. The signature function is in the form

$$\sigma_i = f_{\mathbf{a},x_i}(U_i) = \mathbf{a}^t U_i + x_i \cdot \mathbf{g} = \mathbf{v}_i$$

  and the trapdoor is used to retrieve the value of $U_i \in R^{m\times m}$ by running it against a given $\mathbf{v}_i - x_i \cdot \mathbf{g}$ as described in Section 2.5. U_i is kept secret and can only be obtained with knowledge of the trapdoor.

- $\sigma^* \leftarrow$ **SignEval**$_{prms}(g, ((x_1, \sigma_1), \cdots, (x_N, \sigma_N)))$ Homomorphically computes the signature σ^* for the computation of g on the signed data set $\{(x_i, \sigma_i)\}$. See Section 3.3.1 for how this is computed in practice.

- $\alpha_g \leftarrow$ **Process**$_{prms}(g)$ Homomorphically computes the verification key α_g for function g from **prms**. See Section 3.3.1 for how this is computed in practice.

- **Verify**$_{pk}(\alpha_g, y, \sigma)$ Verify that y really is the result of $g(x_1, \cdots, x_N)$ by checking the signature σ against α_g. In particular, it checks whether

$$f_{\mathbf{a},y}(\sigma^*) = \mathbf{a}^t \sigma^* + y \cdot \mathbf{g} \overset{?}{=} \alpha_g$$

  or, with matrix notation, whether

$$f_{\mathbf{a},y}(U^*) = \mathbf{a}^t U^* + y \cdot \mathbf{g} \overset{?}{=} \mathbf{v}^*$$

Remark 3.3.1. *Since we are working with* truncated polynomials*, we must also ensure that, when evaluating **SignEval** and **Process**, the output is still a polynomial of degree at most $d-1$, or verification correctness will not hold. Any function g whose output would be a polynomial of higher degree is* not an admissible function.

3.3.1 Homomorphic operations

Homomorphic operations for the algorithms **SignEval** and **Process** are quite similar to what already given for the scheme on $\mathbb{Z}_q$. What is different is the noise growth (and the fact that now $x_1, x_2 \in R$, of course). Recall that the norm infinity of a polynomial $||x||_\infty$ is defined as its biggest coefficient.

- **Addition gate:** $g(x_1, x_2) = x_1 + x_2$.

$$U^* = U_1 + U_2 \qquad \mathbf{v}^* = \mathbf{v}_1 + \mathbf{v}_2$$

  The noise of the result is $\beta^* \leq \beta_1 + \beta_2$.

- **Multiplication gate:** $g(x_1, x_2) = x_1 \cdot x_2$.

$$U^* = x_2 \cdot U_1 + U_2\ \mathbf{g}^{-1}(\mathbf{v}_1) \qquad \mathbf{v}^* = \mathbf{v}_2 \cdot G^{-1}(\mathbf{v}_1)$$

  The noise of the result is $\beta^* \leq ||x_2||_\infty \beta_1 + m\beta_2$.

- **Addition-with-constant gate:** $g(x) = x + c$, with $c \in R$.

$$U^* = U \qquad \mathbf{v}^* = \mathbf{v} + (x + c) \cdot \mathbf{g}$$

  The noise of the result is $\beta^* = \beta$, resulting in no change.

- **Multiplication-by-constant gate:** $g(x) = x \cdot c$, with $c \in \mathbb{Z}_q$.

$$U^* = c \cdot U \qquad \mathbf{v}^* = c \cdot \mathbf{v}$$

  The noise of the result is $\beta^* = ||c||_\infty \beta$. Here as well there is also an alternative way, independent of a:

$$U^* = U\ \mathbf{g}^{-1}(a \cdot \mathbf{g}) \qquad \mathbf{v}^* = \mathbf{v}\ \mathbf{g}^{-1}(a \cdot \mathbf{g})$$

  With the noise of the result being $\beta^* = m\beta$.

3.3.2 Improvements over the original scheme

Moving the scheme from the integer moduli to polynomial rings yields several improvements. In this section we go through the differences in the schemes and point out advantages and disadvantages of the scheme on polynomial rings.

It may be helpful to keep in mind that 'signatures secrets', in this context, consist of matrices of polynomials $U \in R^{m \times k}$. One may also think about these as matrices of vectors, i.e. having one extra dimension with respect to the scheme over $\mathbb{Z}_q$.

Signature capabilities. With the scheme over $\mathbb{Z}_q$, only single bits could be signed at a time, requiring a full matrix of size $n \times m$ for each of them. Instead, with polynomials, it is possible to sign a full polynomial of degree d in one shot, resulting in a vector $\mathbf{v} \in R^k$.

A polynomial with coefficients in $\{-1, 0, 1\}$ can be interpreted as a binary vector, which in turn can be read as an integer. Thus, the ability to sign a polynomial results in the ability to sign an integer. In particular, having it degree $d - 1$, it may be possible to sign integers up to $2^d - 1$.

One may wonder how come that switching from integers to polynomials yields this big advantage. Our intuition is that the reason lies in the polynomials being a richer, more complex ring. A binary vector does not belong to $\mathbb{Z}_q$, so it makes sense that signing it with elements of $\mathbb{Z}_q$ requires going through each component individually. Instead, a binary vector does belong to R, and we may sign it wholly.

In other words, by enriching the ring we achieved the ability to sign binary vectors, which are *complex elements* when viewed from the prospective of integers, but *elementary* in the realm of polynomials.

Also, notice how any serious computation in the scheme over $\mathbb{Z}_q$ would require first building the integers from the individual bits through homomorphic operations by evaluating an arithmetic circuit. In other words, to just *build* an integer, one needs to homomorphically evaluate a function on the data set. This overhead is no longer present with polynomials, that already encode full integers.

Error growth. The error grows more quickly in the scheme over polynomials. In fact, since we must take care in always keeping below the maximum degree, we can get to the maximum allowed noise quicker.

Indeed, this is a consequence of the ability of signing full integers (in their binary representation). We *do not have to build the integers through an arithmetic circuit, but we pay the price of being allowed less homomorphic operations.* From a practical point of view, though, this way may be much more handy.

Computation complexity. Moving to polynomials yields differences for computation complexity as well. The scheme on $\mathbb{Z}_q$ essentialy involves several matrix multiplications. To date, the best algorithm for matrix multiplication is the (improved [VW14]) Coppersmith-Winograd algorithm and runs in $O(n^{2.373})$, with n the size of the matrix.

The scheme on polynomials involves essentially the same operations, but the elementary elements are not integers but rather polynomials. Using the Fast Fourier Transform [YB17], it is possible to compute polynomial multiplication in $O(d \log d)$, with d the degree, for an overall rough complexity $O(kd \log d) = O(k \log_2 q \log d)$. This should be comparable to the complexity of evaluating an arithmetic circuit to multiply two integers, which is what we would need to do with the scheme on integers. So computation-wise, we do not seem to be losing anything.

Signature size and parameter choice. The signature size is closely related to the choice of parameters. For integer SIS, we have seen that hardness is closely related to the parameters n and β. In our polynomial settings, parameters governing hardness are the polynomial degree d and the size k of the vector $\mathbf{a}$. We could not find anything in the literature about the choice of parameters for Ring-SIS hardness, but it seems reasonable to require that $n \simeq dk$.

Intuitively, this would mean that the hardness, that was previously embedded only in n, can be spread out on two different sides: the polynomial degree and the vector size. Since the requirement is on their product, this can also give some flexibility over optimization for different scenarios.

Chapter 4

Conclusions

Homomorphic signature schemes have received more and more attention in recent years as the need and usage of cloud services expanded. As we have seen, these schemes allow to delegate the computation on a data set to an untrusted server, with the guarantee that the user will be able to tell whether the final result is correct or not. The way this is achieved is by *signing* the data set, computing a *result signature* through apt operations on the signatures, and then verifying the final signature. Let us also remark that homomorphic signatures schemes have proven useful in other application domains as well, such as electronic voting, smart grids and electronic health records [TDB].

In this work we covered the first scheme to provide *full* homomorphic signatures [GVW15], i.e. allowing the computation of any function, as long as it can be expressed as an arithmetic circuit of pre-defined depth. The scheme is based on lattices, which guarantees strength even with respect to quantum adversaries, and lattice trapdoor functions. We went through the supported homomorphic operations, and saw that they allow to model any arithmetic circuit, as long as the signature *noise* does not exceed some threshold.

The scheme on $\mathbb{Z}_q$ as we presented it, though, has several limitations and elements that make it unpractical for real life applications. In fact, signatures can get very big, only individual bits can be signed, and, most importantly, the verification time depends on the function to be evaluated. In other words, verifying the correctness of a result may require as much time as computing it in the first place. Although we did not provide any concrete improvements in this direction, we proposed some ideas that could speed up the verification process by relaxing the requirements to a *probabilistic-guarantee* on the correctness.

We then showed how to achieve the ability to sign full integers (instead of single bits only), by moving the scheme to a quotient ring R of polynomials with coefficients in $\mathbb{Z}_q$. In this fashion, we lose some *granularity*, in the sense that we cannot operate on the individual bits anymore, but we gain the advantage of working at a *higher level.*

Let us remark that the topic is pretty recent and there is still a lot of work to be done before any homomorphic signature scheme may be implemented and released for mainstream usage. In particular, further work can be done to improve the performance of the scheme to sign whole integers and its storage requirements. Limiting the error growth is another subject for future research that could yield significant benefits.

Appendix: Mathematical references

Definition 4.0.1. *A* group *is a set G together with an operation $+$ such that $(G, +)$ satisfies the given properties:*

- ***Closure**: for all $a, b \in G$, it must hold that $a + b \in G$;*
- ***Associativity**: for all $a, b, c \in G$, it must hold that $(a + b) + c = a + (b + c)$;*
- ***Identity element existence**: there exists an element $e \in G$ such that, for all $a \in G$, it holds that $a + e = e + a = a$;*
- ***Inverse element existence**: for all $a \in G$, there exists an element $b \in G$ such that $a + b = b + a = e$. This b is denoted with a^{-1}.*

Definition 4.0.2. *Given a group $(G, +)$, a subset $H \subset G$ is a* subgroup *of G if the restriction of the operation $+$ to $H \times H$ is a group operation on H (i.e. satisfies the operation properties).*

Definition 4.0.3. *A* unimodular matrix *is a square matrix with entries in $\mathbb{Z}$ having determinant $+1$ or -1.*

Definition 4.0.4. *An* optimization problem *is the problem of finding the best solution (minimum or maximum) from the set of all the possible solutions (where 'possible' means that they satisfy some constraints).*

Definition 4.0.5. *An* euclidean space *is a vector space with a dot product defined over its elements. For example, $\mathbb{R}$ with the standard dot product is an euclidean space.*

Definition 4.0.6. *Given a homomorphism $f : G \to H$, the kernel of f (denoted with $ker(f)$) is the subset of G consisting of all the elements of G that are mapped by f to the identity element of H. That is,*

$$Ker(f) = \{g \in G : f(g) = e_H\}$$

Theorem 4.0.1 (First isomorphism theorem for groups). *Let G and H be groups, and $f : G \to H$ a homomorphism. Then the image of f is isomorphic to the quotient group $G/Ker(f)$.*

Bibliography

[Ajt96] 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. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 99–108. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237838. URL: http://doi.acm.org/10.1145/237814.237838.

[Ajt99] Miklós Ajtai. “Generating Hard Instances of the Short Basis Problem”. In: *Proceedings of the 26th International Colloquium on Automata, Languages and Programming.* ICAL ’99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 1–9. ISBN: 3-540-66224-3. URL: http://dl.acm.org/citation.cfm?id=646229.681554.

[AP11] Joël Alwen and Chris Peikert. “Generating Shorter Bases for Hard Random Lattices”. In: *Theory of Computing Systems* 48.3 (2011), pp. 535–553. ISSN: 1433-0490. DOI: 10.1007/s00224-010-9278-3. URL: https://doi.org/10.1007/s00224-010-9278-3.

[Ber+18] Pauline Bert et al. “Practical Implementation of Ring-SIS/LWE Based Signature and IBE”. In: *Post-Quantum Cryptography.* Ed. by Tanja Lange and Rainer Steinwandt. Cham: Springer International Publishing, 2018, pp. 271–291. ISBN: 978-3-319-79063-3.

[Gen09] Craig Gentry. “A fully homomorphic encryption scheme”. crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. “Trapdoors for Hard Lattices and New Cryptographic Constructions”. In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing.* STOC ’08. Victoria, British Columbia, Canada: ACM, 2008, pp. 197–206. ISBN: 978-1-60558-047-0. DOI: 10.1145/1374376.1374407. URL: http://doi.acm.org/10.1145/1374376.1374407.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. “Leveled Fully Homomorphic Signatures from Standard Lattices”. In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing.* STOC ’15. Portland, Oregon, USA: ACM, 2015, pp. 469–477. ISBN: 978-1-4503-3536-2. DOI: 10.1145/2746539.2746576. URL: \url{http://doi.acm.org/10.1145/2746539.2746576}.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. “NTRU: A Ring-Based Public Key Cryptosystem”. In: *Proceedings of the Third International Symposium on Algorithmic Number Theory*. ANTS-III. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 267–288. ISBN: 3-540-64657-4. URL: http://dl.acm.org/citation.cfm?id=648184.749737.

[Kap04] Eyal Kaplan. *CVP Algorithm*. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/cvp.pdf. [Online; accessed 31-July-2019]. 2004.

[KGV16] A. Khedr, G. Gulak, and V. Vaikuntanathan. “SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers”. In: *IEEE Transactions on Computers* 65.9 (2016), pp. 2848–2858. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2500576.

[LM18] Vadim Lyubashevky and Daniele Micciancio. *On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem*. https://slideplayer.com/slide/13543875. [Online; accessed 31-July-2019]. 2018.

[MP12] Daniele Micciancio and Chris Peikert. “Trapdoors for lattices: Simpler, tighter, faster, smaller”. In: *In EUROCRYPT*. 2012, pp. 700–718.

[MP13] Daniele Micciancio and Chris Peikert. “Hardness of SIS and LWE with small parameters”. In: (Jan. 2013). DOI: 10.1007/978-3-642-40041-4_2.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[Pei09] Chris Peikert. “Public-key Cryptosystems from the Worst-case Shortest Vector Problem: Extended Abstract”. In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC ’09. Bethesda, MD, USA: ACM, 2009, pp. 333–342. ISBN: 978-1-60558-506-2. DOI: 10.1145/1536414.1536461. URL: http://doi.acm.org/10.1145/1536414.1536461.

[Pei16] Chris Peikert. “A Decade of Lattice Cryptography”. In: *Found. Trends Theor. Comput. Sci.* 10.4 (Mar. 2016), pp. 283–424. ISSN: 1551-305X. DOI: 10.1561/0400000074. URL: https://web.eecs.umich.edu/~cpeikert/pubs/lattice-survey.pdf.

[PR06] Chris Peikert and Alon Rosen. “Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices”. In: *Theory of Cryptography*. Ed. by Shai Halevi and Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 145–166. ISBN: 978-3-540-32732-5.

[SD18] Noah Stephens-Davidowitz. *Ring-SIS and Ideal Lattices*. https://people.csail.mit.edu/vinodv/6876-Fall2018/RingSISclass.pdf. [Online; accessed 31-July-2019]. 2018.

[Sho82] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM JOURNAL ON COMPUTING* 26 (1982), pp. 1484–1509.

[TDB] Giulia Traverso, Denise Demirel, and Johannes Buchmann. *Homomorphic Signature Schemes - A Survey*. https://pdfs.semanticscholar.org/0d25/419e80fff2599c47d5c86f77ebe717feb534.pdf. [Online; accessed 31-July-2019].

[Unk11] Unknown. *Lattices and Cryptology.* `https://www.isical.ac.in/~shashank_r/lattice.pdf`. [Online; accessed 31-July-2019]. 2011.

[VW14] Virginia Vassilevska Williams. “Breaking the Coppersmith-Winograd barrier”. In: (Sept. 2014).

[YB17] Jia Yan-Bin. *Polynomial Multiplication and Fast Fourier Transform.* `http://web.cs.iastate.edu/~cs577/handouts/polymultiply.pdf`. [Online; accessed 31-July-2019]. 2017.

[Mer] Merriam Webster Dictionary. *Homomorphism.* `https://www.merriam-webster.com/dictionary/homomorphism`. [Online; accessed 31-July-2019].

[Wik19a] Wikipedia contributors. *Gram-Schmidt Process — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/wiki/Gram-Schmidt_process`. [Online; accessed 31-July-2019]. 2019.

[Wik19b] Wikipedia contributors. *Lenstra–Lenstra–Lovasz lattice basis reduction algorithm — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/wiki/Lenstra-Lenstra-Lovasz_lattice_basis_reduction_algorithm`. [Online; accessed 31-July-2019]. 2019.